https://blog.csdn.net/hfcenter/article/details/70212951

**Log4j2的简要使用**

2017年04月17日 16:17:28 [xwhfcenter](https://me.csdn.net/hfcenter) 阅读数：2774 标签： [log4j2](http://so.csdn.net/so/search/s.do?q=log4j2&t=blog)[properties](http://so.csdn.net/so/search/s.do?q=properties&t=blog)[xml](http://so.csdn.net/so/search/s.do?q=xml&t=blog) 更多

个人分类： [研发之路](https://blog.csdn.net/hfcenter/article/category/6867139)

版权声明：本文为博主原创文章，未经博主允许不得转载。 https://blog.csdn.net/hfcenter/article/details/70212951

* 刚好这周的一项任务是把Druid平台的Log4j2的xml配置文件翻译成properties，原因是xml格式的配置文件看着不舒服
* 在网上查了下资料，Log4j进入2.x版本后，一开始就抛弃了properties格式的配置文件，到2.4版本的时候才又重新支持，猜测是Log4j1.x时代养成了用户的习惯，不好改了，所以发觉这任务还有点意义
* 本文写作的时候Log4j2已经发布到2.8.2的版本了，但是Druid用的是2.5，所以按照2.5的规则来配置，ps: 2.x版本的properties的语法和1.x的差别很大
* 没有完美的资料，即使是官网也没能讲全properties的语法规则，只能靠自己灵光咋现的顿悟一点点爬行

## Log4j2配置文件的读取顺序

官网上给了一串的if-else的说明，概括起来就是一条读取顺序链

System Property -> properties -> YAML -> JSON -> XML -> Default Configuration

* YAML配置文件包括: .yaml和.yml
* JSON配置文件包括: .json和.jsn

## Appender和Logger说明

### Appender说明

Appenders are responsible for delivering LogEvents to their destination.

简单理解一下，就是决定Log4j2产生的日志要发到什么地方   
我们设置了两处用来保留Druid的日志，一个是服务器集群的本地文件，另一个是Kafka

### Appender配置

#### 基本配置

# 采用RollingFile类型的Appender

appender.rolling.type = RollingFile

# 本appender的名字，以便在Logger的配置项中能够调用

appender.rolling.name = broker

# 当前正在操作的日志文件的文件名

appender.rolling.fileName = /druid/log/broker.log

# 归档后的日志文件的文件名格式，其中`%d{yyyy-MM-dd-HH}`用来自动填充日期

appender.rolling.filePattern = /druid/log/broker-%d{yyyy-MM-dd-HH}.log.gz

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8

#### Layout配置

appender.rolling.layout.type = PatternLayout

# 对应输出的每条日志的日志头

appender.rolling.layout.pattern = %date- %c{2}: %m%n

* 1
* 2
* 3

#### Policy配置

appender.rolling.policies.type = Policies

# 基于时间进行日志的切割

appender.rolling.policies.time.type = TimeBasedTriggeringPolicy

# 切割的间隔为1小时, 即每小时进行一次日志的归档

appender.rolling.policies.time.interval = 1

# 修正时间范围, 从0时开始计数

appender.rolling.policies.time.modulate = true

* 1
* 2
* 3
* 4
* 5
* 6
* 7

除了时间之外还可以基于文件的大小进行切割，相关的策略类为SizeBasedTriggeringPolicy

#### Strategy配置

配置Strategy用来删除过多的日志归档文件

appender.rolling.strategy.type = DefaultRolloverStrategy

appender.rolling.strategy.delete.type = Delete

# 删除哪个目录下的日志归档文件

appender.rolling.strategy.delete.basePath = /druid/log

# 从basePath起向下遍历几级子文件夹？

appender.rolling.strategy.delete.maxDepth = 1

appender.rolling.strategy.delete.iffile.type = IfFileName

# 删除符合该模式文件名的归档日志文件

appender.rolling.strategy.delete.iffile.glob = broker-\*.log.gz

appender.rolling.strategy.delete.iflastmodify.type = IfLastModified

# 保留多少天的日志？

appender.rolling.strategy.delete.iflastmodify.age = 2d

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12

#### KafkaAppender的额外配置

appender.kafka.type = Kafka

appender.kafka.topic = druid-log

appender.kafka.brokers = <brokerIp>:9092

* 1
* 2
* 3

因为propertie文件的弱表达性不支持Kafka的bootstrap.servers这种带有.的属性，所以对Loger4j2中的KafkaManager类做了一点修改，改成了通过broker属性获取值，然后再转换成bootstrap.servers写入Kafka配置

### Logger说明

* Logger的配置用来细节化地配置具体类的日志输出选项
* Logger中必须要配置的是root级别的Logger，用rootLogger作为属性
* 其他的Logger由用户自行定义

# 自定义的logger名字

logger.emitter.name = com.metamx.emitter.core.LoggingEmitter

# 该logger的日志级别

logger.emitter.level = info

logger.emitter.additivity = true

# 该logger通过什么类型的appender来输出

logger.emitter.appenderRefs = kafka

# 该logger通过该类型具体的哪个appender来输出

logger.emitter.appenderRef.kafka.ref = kafka-monitor

# root的配置和自定义Logger一样

rootLogger.level = info

rootLogger.appenderRefs = rolling

rootLogger.appenderRef.rolling.ref = middleManager

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14

## XML怎么翻译成properties

* 这个钻研了很久，官方的文档翻阅了很多遍，只能看出一些端倪来
* Google的结果基本只有一两篇有价值的
* 读了一小部分Log4j2的源代码，了解了propertie文件的解析方式
* 最后终于在各种连蒙带猜中，顿悟除了规则来

## properties配置文件读取

Log4j-2.5对properties文件的解析位于PropertiesConfigurationFactory类中

1. 解析appenders键、loggers键，获得所有的appender和logger
2. rootLogger键是另外单独解析的
3. 为每个appender调用createAppender()方法，为每个logger调用createLogger()方法
4. properties配置文件的键是根据符合.来进行区分的，一个.隔开了上下层的级联，在createAppender()方法和createLogger()方法中，就是一层一层的递归遍历来获取有级联关系的配置
5. 从代码中可以看出, 在appender中每配置一个组件都要先定义一个type，然后就可以在这一层定义属性键 = 对应的值

遍历配置的代码如下, 不包括一些默认的属性键

private void processRemainingProperties(ComponentBuilder<?> builder, String name, Properties properties) {

while (properties.size() > 0) {

String propertyName = properties.stringPropertyNames().iterator().next();

int index = propertyName.indexOf('.');

if (index > 0) {

String prefix = propertyName.substring(0, index);

Properties componentProperties = PropertiesUtil.extractSubset(properties, prefix);

builder.addComponent(createComponent(builder, prefix, componentProperties));

} else {

builder.addAttribute(propertyName, properties.getProperty(propertyName));

properties.remove(propertyName);

}

}

}

* 1
* 2
* 3
* 4
* 5
* 6
* 7
* 8
* 9
* 10
* 11
* 12
* 13
* 14
* 15

## xml文件与properties文件的对应

一开始没能理解官网给的例子，后来顿悟的关键点就在这个type上

* xml类似于HTML中的标签写法，所以**每个标签名就对应properties文件中的type**
* 每个标签中跟进的属性就可以转化成properties文件中该type层下的键值对

举个例子， 有appender的切割策略配置如下:

<Policies>

<TimeBasedTriggeringPolicy intervel = "2" modulate="true"/>

<SizeBasedTriggeringPolicy size="100 MB"/>

</Policies>

* 1
* 2
* 3
* 4

# 对应Policies标签

appender.rolling.policies.type = Policies

# 对应TimeBasedTriggeringPolicy

appender.rolling.policies.time.type = TimeBasedTriggeringPolicy

appender.rolling.policies.time.interval = 2

appender.rolling.policies.time.modulate = true

# 对应SizeBasedTriggeringPolicy

appender.rolling.policies.size.type = SizeBasedTriggeringPolicy

appender.rolling.policies.size.size=100MB