Logging

Certified Developer
The Ultimate Tech Degree

DigitalHouse >
Coding School

O que precisamos para gerar log?

Para gerar logs, precisamos adicionar um arquivo de configuração. Vamos adicioná-lo na raiz do projeto, ou seja, na pasta principal. Por exemplo, se nosso projeto tem o nome app, nessa pasta criamos um arquivo de configuração com o nome **log4j.properties** com o seguinte conteúdo:

O que precisamos para gerar log?

Na primeira linha, estamos indicando o nível mínimo de logging (registro) e os appenders que vamos usar. Nesse caso, usaremos um nível de logging definido como DEBUG e criaremos dois appenders, stdout e file.

```
log4j.rootLogger=DEBUG, stdout, file
```

A segunda linha é usada para configurar em que nível os avisos começarão a ser exibidos pelo console e armazenados no arquivo.

```
log4j.logger.infoLogger=DEBUG
```

O que precisamos para gerar log?

E, com a terceira linha, evitamos que os appenders herdem a configuração de seus appenders pais, se houver algum (no nosso, seria o appender principal, então não temos esse problema).

```
log4j.additivity.infoLogger = false
```

Criar a configuração para imprimir mensagens por console

Na primeira linha indicamos que tipo de logger será, referindo-se à classe que irá imprimir as mensagens (Lembre-se da seção **Appenders**!).

```
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
```

Na segunda linha, dizemos que queremos imprimí-lo diretamente do console.

```
log4j.appender.stdout.Target=System.out
```

Criar a configuração para imprimir mensagens por console

E as duas últimas linhas são para configurar o template que cada mensagem terá. Você pode ver todas as opções de configuração possíveis na página de ajuda do Apache log4J

```
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.stdout.layout.ConversionPattern=[%d{yyyy-MM-dd HH:mm:ss}] [ %-5p] [%c{1}:%L] %m%n
```

A mensagem de saída seria algo assim:

```
[2018-08-03 11:48:39] [ INFO ] [App:29] Este é um teste a partir do App class
```

Configurar o appender

Nessas linhas, vamos fazer exatamente o mesmo que antes, mas configurando o appender para a saída por meio de um arquivo.

Na primeira linha configuramos a classe como RollingFileAppender, o que significa que diferentes arquivos serão criados quando certas condições com as quais lidaremos nas linhas a seguir forem atendidas.

```
log4j.appender.file=org.apache.log4j.RollingFileAppender
```

Na próxima linha, indicamos o nome (com o caminho/rota incluído) que queremos que nosso arquivo de log tenha.

```
log4j.appender.file.File=avisos.log
```

Configurar o appender

Com MaxFileSize estabelecemos o tamanho máximo que nosso arquivo terá, e com MaxBackupIndex indicamos quantos arquivos podemos ter usando o mesmo log. Ao atingir o máximo, eles começarão a ser sobrescritos começando pelos mais antigos.

```
log4j.appender.file.MaxFileSize=5MB
```

E, finalmente, como no console, indicamos qual modelo (template) nossas mensagens terão.

```
log4j.appender.file.layout=org.apache.log4j.PatternLayout
```

```
log4j.appender.file.layout.ConversionPattern=[%d{yyyy-MM-dd
HH:mm:ss}] [ %-5p] [%c{1}:%L] %m%n
```

Conclusão

A classe que vimos nos dá a possibilidade de gerar logs em diferentes tipos de registro, que vão de mãos dadas com o tipo de informação que queremos mostrar.

Os tipos de logs nos fornecem informações mais exatas sobre o que está acontecendo, por exemplo, se usarmos:

```
logger.info("Esta é uma mensagem apenas informativa");
```

Vemos que só nos ajuda a adicionar um comentário sobre o que aconteceu naquele momento, portanto, não nos agrega muito valor.

Conclusão

Por outro lado, se adicionamos algo como:

```
Logger.error("O usuário é inválido", e);
```

Claramente o log nos disse que temos um erro, e a letra ‘e’ como segundo parâmetro nos vai imprimir no log, o stacktrace do erro. Um exemplo seria:

```
Exception in thread "main" java.lang.NullPointerException at
com.example.myproject.Book.getTitle(Book.java:16) at
com.example.myproject.Author.getBookTitles(Author.java:25) at
com.example.myproject.Bootstrap.main(Bootstrap.java:14)
```

Podemos observar que o erro principal foi NullPointerException e vemos onde foi gerado.

Conclusão

Níveis de Registro

Como vimos anteriormente, os logs levels (níveis de registro) são: FATAL, ERROR, WARN, INFO, DEBUG, TRACE y ALL.

Agora podemos ver a visibilidade, dependendo de qual escolhermos em:

```
log4j.logger.infoLogger=DEBUG
```

Níveis de registro (X=Visible)

	FATAL	ERROR	WARN	INFO	DEBUG	TRACE	ALL
FATAL	X						
ERROR	X	X					
WARN	X	X	X				
INFO	X	X	X	X			
DEBUG	X	X	X	X	X		
TRACE	X	X	X	X	X	X	
ALL	X	X	X	X	X	X	X


DigitalHouse>
Coding School