Red Hat build of Quarkus 3.8

日志记录配置

Last Updated: 2024-06-24

法律通告

摘要

阅读红帽构建的 Quarkus 中使用日志 API，配置日志输出，以及使用日志记录适配器进行统一输出。

目录

提供有关红帽构建的 QUARKUS 文档的反馈

要报告错误或改进文档，请登录到 Red Hat JIRA 帐户并提交问题。如果您没有 Red Hat Jira 帐户，则会提示您创建一个帐户。

步骤

1. 单击以下链接 **以创建 ticket**。
2. 在 **Summary** 中输入问题的简短描述。
3. 在 **Description** 中提供问题或功能增强的详细描述。包括一个指向文档中问题的 URL。
4. 点 **Submit** 创建问题，并将问题路由到适当的文档团队。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中有问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。详情请查看 CTO Chris Wright 的信息。

第 1 章 日志记录配置

阅读在 Quarkus 中使用日志 API，配置日志输出，以及使用日志记录适配器统一其他日志记录 API 的输出。

Quarkus 使用 JBoss Log Manager 日志记录后端发布应用程序和框架日志。Quarkus 支持 JBoss Logging API 和多个其他日志记录 API，与 JBoss Log Manager 无缝集成。您可以使用以下 API 中的任何一种 ：

- JBoss Logging
- JDK **java.util.logging** (JUL)
- SLF4J
- Apache Commons Logging
- Apache Log4j 2
- Apache Log4j 1

1.1. 使用 JBOSS LOGGING 进行应用程序日志记录

使用 JBoss Logging API 时，您的应用程序不需要额外的依赖项，因为 Quarkus 会自动提供它。

使用 JBoss Logging API 记录消息的示例：

```
import org.jboss.logging.Logger;

import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import jakarta.ws.rs.Produces;
import jakarta.ws.rs.core.MediaType;

@Path("/hello")
public class ExampleResource {

    private static final Logger LOG = Logger.getLogger(ExampleResource.class);

    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String hello() {
        LOG.info("Hello");
        return "hello";
    }
}
```

注意

虽然 JBoss Logging 直接将日志消息路由到 JBoss Log Manager，但您的库之一可能依赖不同的日志记录 API。在这种情况下，您需要使用 日志适配器 来确保其日志消息被路由到 JBoss Log Manager。

1.2. 获取应用日志记录器

在 Quarkus 中，获取应用程序日志记录器的最常见方法是：

- 声明日志记录器字段
- 简化的日志记录
- 注入配置的日志记录器

1.2.1. 声明日志记录器字段

借助此类方法，您可以使用特定的 API 获取日志记录器实例，将其存储在类的静态字段中，并在此实例上调用日志记录操作。

同一流可以使用 任何受支持的日志记录 API 应用。

使用 JBoss Logging API 将日志记录器实例存储在静态字段中的示例：

```
package com.example;

import org.jboss.logging.Logger;

public class MyService {
    private static final Logger log = Logger.getLogger(MyService.class); (1)

    public void doSomething() {
        log.info("It works!"); (2)
    }
}
```

1. 定义 logger 字段。
2. 在 **log** 对象上调用所需的日志记录方法。

1.2.2. 简化的日志记录

Quarkus 通过自动将日志记录器字段添加到使用 **io.quarkus.logging.Log** 的类来简化日志记录。这消除了重复板代码的需求，并增强了日志记录设置便利。

使用静态方法调用的简化日志示例：

```
package com.example;

import io.quarkus.logging.Log; (1)

class MyService { (2)
    public void doSomething() {
        Log.info("Simple!"); (3)
    }
}
```

1. **io.quarkus.logging.Log** 类包含与 JBoss Logging 相同的方法，但它们 **是静态的**。
2. 请注意，该类没有声明 logger 字段。这是因为在应用程序构建过程中，使用 **Log** API 的每个类中会自动创建 **私有静态最终 org.jboss.logging.Logger** 字段，调用 **Log** 方法的类的完全限定名称被用

自动创建 **私有静态最终 org.jboss.logging.Logger** 字段。调用 **Log** 方法的类的完全限定名称被用作日志记录器名称。在本例中，日志记录器名称将是 **com.example.MyService**。

3 最后，在应用程序构建过程中，所有对 **日志** 方法的调用都会重写为常规的 JBoss Logging 调用。

警告

仅在应用程序类中使用 **Log** API，而不是在外部依赖项中使用。构建时没有由 Quarkus 处理的日志方法调用将抛出异常。

1.2.3. 注入配置的日志记录器

使用 @Inject 注解配置的 org.jboss.logging.Logger 日志记录器实例的注入是添加应用程序日志记录器的另一个替代方法，但仅适用于 CDI Bean。

您可以使用 @Inject Logger 日志，其中日志记录器以您注入的类命名，或 @Inject @LoggerName ("...") Logger log，其中日志记录器将接收指定名称。注入后，您可以使用 log 对象来调用日志记录方法。

两种不同类型的日志记录器注入示例：

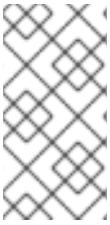
```
package com.example;

import org.jboss.logging.Logger;

@ApplicationScoped
class SimpleBean {

  @Inject
  Logger log; (1)

  @LoggerName("foo")
  Logger fooLog; (2)

  public void ping() {
    log.info("Simple!");
    fooLog.info("Goes to _foo_ logger!");
  }
}
```

1

声明类的 FQCN 用作日志记录器名称，例如 org.jboss.logging.Logger.getLogger (SimpleBean.class)。

2

在这种情况下，名称 *foo* 用作日志记录器名称，例如org.jboss.logging.Logger.getLogger ("foo")。



注意

日志记录器实例在内部缓存。因此，当注入日志记录器（例如，在 @RequestScoped bean 中）时，它会共享所有 bean 实例，以避免可能与日志记录器实例化相关的性能。

1.3. 使用日志级别

Quarkus 提供不同的日志级别，这有助于开发人员根据事件的严重性控制记录的信息量。

表 1.1. Quarkus 使用的日志级别

OFF	用于配置的特殊级别，以关闭日志记录。
FATAL	关键服务故障或无法满足任何类型的请求。
ERROR	请求出现重大中断，或者无法服务请求。
WARN	可能需要立即修正的非关键服务错误或问题。
INFO	服务生命周期事件或重要相关的低频率信息。
DEBUG	提供有关生命周期或非绑定事件的额外信息的消息，可用于调试。
TRACE	传递额外每个请求的调试信息的消息可能非常高。
ALL	在配置中使用特殊级别来打开所有消息的日志记录，包括自定义级别。

您还可以为使用 java.util.logging 的应用程序和库配置以下级别：

严重	与 **ERROR** 相同。
WARNING	与 **WARN** 相同。
CONFIG	服务配置信息。
FINE	与 **DEBUG** 相同。
FINER	与 **TRACE** 相同。
FINEST	与 **TRACE** 相比增加了调试输出，这可能会更高频率。

表 1.2. 级别之间的映射

数字级别值	标准级别名称	等效的 java.util.logging (JUL) 级别名称
1100	FATAL	Not applicable
1000	ERROR	严重
900	WARN	WARNING
800	INFO	INFO
700	Not applicable	CONFIG
500	DEBUG	FINE
400	TRACE	FINER
300	Not applicable	FINEST

1.4. 配置日志级别、类别和格式

JBoss Logging 集成到 Quarkus 中，通过设置所有可用扩展的单一配置文件，为 所有支持的日志记录 API 提供统一配置。要调整运行时日志记录，请修改 application.properties 文件。

有关如何将默认日志级别设置为 INFO 日志的示例，并包含 Hibernate DEBUG 日志：

```
quarkus.log.level=INFO
quarkus.log.category."org.hibernate".level=DEBUG
```

当您将日志级别设置为低于 DEBUG 时，您还必须调整最小日志级别。此设置可以是全局的，使用 quarkus.log.min-level 配置属性或每个类别：

```
quarkus.log.category."org.hibernate".min-level=TRACE
```

这会设置 Quarkus 需要为其生成支持代码的 floor 级别。构建时必须设置最小日志级别，以便 Quarkus 能够打开一项操作，以优化可以在不可用级别上登录的机会。

来自原生执行的示例：

将 INFO 设置为最低日志记录级别，将 isTraceEnabled 等低级别检查设置为 false。这标识了诸如 if (logger.isDebug （）） callMethod （） 的代码； 该代码永远不会被执行并将其标记为"dead"。

警告

如果您在命令行中添加这些属性，请确保正确转义 " 字符：

```
-Dquarkus.log.category.\"org.hibernate\".level=TRACE
```

所有潜在的属性都列在 日志记录配置参考部分。

1.4.1. 日志记录类别

日志记录会根据每个类别进行配置，每个类别都独立配置。除非有更具体的子类别配置，否则类别的配置会以递归方式适用于所有子类别。

所有日志记录类别的父级称为"root 类别"。作为最终的父级，此类别可能包含全局应用到所有其他类别的配置。这包括全局配置的处理程序和格式器。

例 1.1. 适用于所有类别的全局配置示例：

```
quarkus.log.handlers=console,mylog
```

在本例中，根类别配置为使用两个处理程序： console 和 mylog。

例 1.2. 一个 per-category 配置示例：

```
quarkus.log.category."org.apache.kafka.clients".level=INFO
quarkus.log.category."org.apache.kafka.common.utils".level=INFO
```

本例演示了如何在 org.apache.kafka. clients 和 org.apache.kafka.common.utils 类别上配置最小日志级别。

如需更多信息，请参阅 日志记录配置参考。

如果要为特定类别配置一些额外内容，请创建一个命名处理程序，如 quarkus.log.handler.[console|file|syslog].<your-handler-name>，并使用 quarkus.log.category.<my-category>.handlers 为该类别设置它。

示例用例可能需要对保存到文件（而不是用于其他处理程序的格式）的日志消息使用不同的时间戳格式。

有关进一步演示，请参阅 Attaching named handlers to a category example。

属性名称	default	描述
quarkus.log.category."<category-name>".level	INFO [a]	配置名为 <category-name> 的类别的级别。引号是必需的。
quarkus.log.category."<category-name>".min-level	DEBUG	配置名为 <category-name> 的类别的最小日志记录级别。引号是必需的。
quarkus.log.category."<category-name>".use-parent-handlers	true	指定此日志记录器是否应该将其输出发送到其父日志记录器。

属性名称	default	描述
quarkus.log.category."<category-name>".handlers=[<handler>]	**empty [b]**	要附加到特定类别的处理程序的名称。

[a] 有些扩展可能会为某些类别定义自定义的默认日志级别，以默认减少日志 noise。在配置中设置日志级别将覆盖任何扩展定义的日志级别。

[b] 默认情况下，配置的类别获取与根日志记录器上附加的处理程序相同的处理程序。

注意

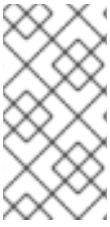
. 符号分隔配置属性中的特定部分。属性名称中的引号用作所需的转义来保持类别规格，如 quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE, intact。

1.4.2. 根日志记录器配置

根日志记录器类别是单独处理的，并使用以下属性进行配置：

属性名称	default	描述
quarkus.log.level	**INFO**	每个日志类别的默认日志级别。
quarkus.log.min-level	**DEBUG**	每个日志类别的默认最小日志级别。

- **如果给定日志记录器类别不存在级别配置，则会检查父类别。**
- **如果没有为类别及其任何父类别提供特定配置，则使用根日志记录器配置。**



注意

虽然根日志记录器的处理程序通常通过 quarkus.log.console、quarkus.log.file 和 quarkus.log.syslog 直接配置，但它无法使用 quarkus.log.handlers 属性附加到它。

1.5. 日志记录格式

Quarkus 使用基于模式的日志格式器，它默认生成人类可读的文本日志，但您也可以使用专用属性为每个日志处理程序配置格式。

对于控制台处理程序，属性为 quarkus.log.console.format。

日志记录格式字符串支持以下符号：

符号	概述	描述
%%	**%**	呈现简单的 **%** 字符。
%c	类别	呈现类别名称。
%C	源类	呈现源类名称。[a]
%d{xxx}	Date	使用给定日期格式字符串呈现日期，它使用 **java.text.SimpleDateFormat** 定义的语法。
%e	例外	呈现出的异常（若有）。
%F	源文件	呈现源文件名称。[a]
%h	主机名	呈现系统简单主机名。
%H	合格主机名	呈现系统的完全限定主机名，其名称可能与简单主机名相同，具体取决于操作系统配置。
%i	进程 ID	呈现当前进程 PID。
%L	源位置	呈现源位置信息，其中包括源文件名、行号、类名称和方法名称。[a]
%L	源行	呈现源行号。[a]
%m	完整消息	呈现日志消息加上例外（如果有）。
%M	源方法	呈现源方法名称。[a]
%n	newline	呈现特定于平台的行分隔符字符串。

符号	概述	描述
%N	进程名称	呈现当前进程的名称。
%p	级别	呈现消息的日志级别。
%r	相对时间	以毫秒为单位呈现应用程序日志开始的时间。
%s	简单消息	只呈现日志消息，没有异常追踪。
%t	线程名称	呈现线程名称。
%T{id}	线程 ID	呈现线程 ID。
%z{<zone name>}	时区	将输出的时区设置为 < **zone name&** gt;。
%x{<MDC 属性 name>}	映射诊断上下文值	呈现来自映射诊断上下文的值。
%X	映射诊断上下文值	呈现 Mapped Diagnostic Context 中的所有值，格式为 **{property.key=property.value}**。
%x	嵌套诊断上下文值	呈现 Nested Diagnostics Context 中的所有值，格式为 **{value1.value2}**。
[a] 检查调用者信息的格式序列可能会影响性能		

1.5.1. 其他控制台日志记录格式

更改控制台日志格式很有用，例如，当 Quarkus 应用的控制台输出被处理并存储日志信息以便稍后进行分析时。

1.5.1.1. JSON 日志记录格式

可使用 quarkus-logging-json 扩展来添加对 JSON 日志记录格式及其相关配置的支持。

1. **在构建文件中添加此扩展，如以下代码片段所示：**

- **使用 Maven：**

```
<dependency>
  <groupId>io.quarkus</groupId>
  <artifactId>quarkus-logging-json</artifactId>
</dependency>
```

- 使用 Gradle ：

```
implementation("io.quarkus:quarkus-logging-json")
```

默认情况下，存在此扩展会替换控制台配置中的输出格式配置，格式字符串和颜色设置（若有）将被忽略。其他控制台配置项目（包括控制异步日志记录和日志级别）将继续应用。

在某些情况下，在 dev 模式中使用人类可读的（不结构化）登录，在生产环境模式中使用 JSON 日志记录(structured)会有意义。这可以通过使用不同配置集来实现，如以下配置所示。

2. 在 dev 和 test 模式的 application.properties 中禁用 JSON 日志记录：

```
%dev.quarkus.log.console.json=false
%test.quarkus.log.console.json=false
```

1.5.1.1.1. 配置

使用支持的属性配置 JSON 日志扩展，以自定义其行为。

构建时修复的配置属性 - 所有其他配置属性可在运行时覆盖

Console logging	类型	default
quarkus.log.console.json 确定是否启用 JSON 控制台格式扩展，它会禁用"常规"控制台格式。 环境变量： **QUARKUS_LOG_CONSOLE_JSON**	布尔值	**true**

quarkus.log.console.json.pretty-print 启用 JSON 记录的"pretty 打印"。请注意，一些 JSON 解析器将无法读取用户友善打印的输出。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_PRETTY_PRINT**	布尔值	**false**
quarkus.log.console.json.date-format 要使用的日期格式。特殊字符串 "default" 表示应使用默认格式。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_DATE_FORMAT**	string	**default**
quarkus.log.console.json.record-delimiter 要使用的特殊记录分隔符。默认情况下使用 newline。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_RECORD_DELIMITER**	string	
quarkus.log.console.json.zone-id 要使用的区域 ID。特殊字符串 "default" 表示应使用默认区域。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_ZONE_ID**	string	**default**
quarkus.log.console.json.exception-output-type 要指定的异常输出类型。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_EXCEPTION_OUTPUT_TYPE**	**详细、格式化、详细和格式化**	详细
quarkus.log.console.json.print-details 在日志中启用打印更多详细信息。 打印详细信息可能比较昂贵，因为值是从调用者检索的。详情包括源类名称、源文件名、源方法名称和源行号。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_PRINT_DETAILS**	布尔值	**false**
quarkus.log.console.json.key-overrides 使用自定义值覆盖键。省略这个值表示不会应用任何键覆盖。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_KEY_OVERRIDES**	string	
quarkus.log.console.json.excluded-keys 要排除在 JSON 输出中的密钥。 环境变量：**QUARKUS_LOG_CONSOLE_JSON_EXCLUDED_KEYS**	字符串列表	

quarkus.log.console.json.additional-field."field-name".value 其他字段值。 环境变量： **QUARKUS_LOG_CONSOLE_JSON_ADDITIONAL_FIELD__FIELD_NAME_VALUE**	string	必需 
quarkus.log.console.json.additional-field."field-name".type 其他字段类型规格。支持的类型：**字符串**、**int** 和 **long**。若未指定，则字符串为默认值。 环境变量： **QUARKUS_LOG_CONSOLE_JSON_ADDITIONAL_FIELD__FIELD_NAME_TYPE**	**字符串,int,long**	**string**
File logging	**类型**	**default**
quarkus.log.file.json 确定是否启用 JSON 控制台格式扩展，它会禁用"常规"控制台格式。 环境变量：**QUARKUS_LOG_FILE_JSON**	布尔值	**true**
quarkus.log.file.json.pretty-print 启用 JSON 记录的"pretty 打印"。请注意，一些 JSON 解析器将无法读取用户友善打印的输出。 环境变量：**QUARKUS_LOG_FILE_JSON_PRETTY_PRINT**	布尔值	**false**
quarkus.log.file.json.date-format 要使用的日期格式。特殊字符串 "default" 表示应使用默认格式。 环境变量：**QUARKUS_LOG_FILE_JSON_DATE_FORMAT**	string	**default**
quarkus.log.file.json.record-delimiter 要使用的特殊记录分隔符。默认情况下使用 newline。 环境变量：**QUARKUS_LOG_FILE_JSON_RECORD_DELIMITER**	string	
quarkus.log.file.json.zone-id 要使用的区域 ID。特殊字符串 "default" 表示应使用默认区域。 环境变量：**QUARKUS_LOG_FILE_JSON_ZONE_ID**	string	**default**

quarkus.log.file.json.exception-output-type 要指定的异常输出类型。 环境变量：**QUARKUS_LOG_FILE_JSON_EXCEPTION_OUTPUT_TYPE**	**详细、格式化、详细和格式化**	**详细**
quarkus.log.file.json.print-details 在日志中启用打印更多详细信息。 打印详细信息可能比较昂贵，因为值是从调用者检索的。详情包括源类名称、源文件名、源方法名称和源行号。 环境变量：**QUARKUS_LOG_FILE_JSON_PRINT_DETAILS**	布尔值	**false**
quarkus.log.file.json.key-overrides 使用自定义值覆盖键。省略这个值表示不会应用任何键覆盖。 环境变量：**QUARKUS_LOG_FILE_JSON_KEY_OVERRIDES**	string	
quarkus.log.file.json.excluded-keys 要排除在 JSON 输出中的密钥。 环境变量：**QUARKUS_LOG_FILE_JSON_EXCLUDED_KEYS**	字符串列表	
quarkus.log.file.json.additional-field."field-name".value 其他字段值。 环境变量： **QUARKUS_LOG_FILE_JSON_ADDITIONAL_FIELD__FIELD_NAME_VALUE**	string	必需
quarkus.log.file.json.additional-field."field-name".type 其他字段类型规格。支持的类型：**字符串**、**int** 和 **long**。若未指定，则字符串为默认值。 环境变量： **QUARKUS_LOG_FILE_JSON_ADDITIONAL_FIELD__FIELD_NAME_TYPE**	**字符串,int,long**	**string**
Syslog 日志记录	**类型**	**default**
quarkus.log.syslog.json 确定是否启用 JSON 控制台格式扩展，它会禁用"常规"控制台格式。 环境变量：**QUARKUS_LOG_SYSLOG_JSON**	布尔值	**true**

quarkus.log.syslog.json.pretty-print 启用 JSON 记录的"pretty 打印"。请注意，一些 JSON 解析器将无法读取用户友善打印的输出。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_PRETTY_PRINT**	布尔值	**false**
quarkus.log.syslog.json.date-format 要使用的日期格式。特殊字符串 "default" 表示应使用默认格式。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_DATE_FORMAT**	string	**default**
quarkus.log.syslog.json.record-delimiter 要使用的特殊记录分隔符。默认情况下使用 newline。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_RECORD_DELIMITER**	string	
quarkus.log.syslog.json.zone-id 要使用的区域 ID。特殊字符串 "default" 表示应使用默认区域。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_ZONE_ID**	string	**default**
quarkus.log.syslog.json.exception-output-type 要指定的异常输出类型。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_EXCEPTION_OUTPUT_TYPE**	**详细、格式化、详细和格式化**	详细
quarkus.log.syslog.json.print-details 在日志中启用打印更多详细信息。 打印详细信息可能比较昂贵，因为值是从调用者检索的。详情包括源类名称、源文件名、源方法名称和源行号。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_PRINT_DETAILS**	布尔值	**false**
quarkus.log.syslog.json.key-overrides 使用自定义值覆盖键。省略这个值表示不会应用任何键覆盖。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_KEY_OVERRIDES**	string	
quarkus.log.syslog.json.excluded-keys 要排除在 JSON 输出中的密钥。 环境变量：**QUARKUS_LOG_SYSLOG_JSON_EXCLUDED_KEYS**	字符串列表	

quarkus.log.syslog.json.additional-field."field-name".value 其他字段值。 环境变量： **QUARKUS_LOG_SYSLOG_JSON_ADDITIONAL_FIELD__FIELD_NAME_VALUE**	string	必需
quarkus.log.syslog.json.additional-field."field-name".type 其他字段类型规格。支持的类型：**字符串**、**int** 和 **long**。若未指定，则字符串为默认值。 环境变量： **QUARKUS_LOG_SYSLOG_JSON_ADDITIONAL_FIELD__FIELD_NAME_TYPE**	**字符串,int,long**	**string**

警告

启用用户用户打印可能会导致某些处理器和 JSON 解析器失败。

注意

打印详细信息可能比较昂贵，因为值是从调用者检索的。详情包括源类名称、源文件名、源方法名称和源行号。

1.6. 日志处理程序

日志处理程序是一个日志记录组件，负责将日志事件委托给接收者。Quarkus 包括几个不同的日志处理程序： 控制台、file 和 syslog。

特色示例使用 com.example 作为日志记录类别。

1.6.1. 控制台日志处理程序

控制台日志处理程序默认为启用，它会将所有日志事件定向到应用程序的控制台，通常是系统的 stdout。

- 全局配置示例：

```
quarkus.log.console.format=%d{yyyy-MM-dd HH:mm:ss} %-5p [%c] (%t) %s%e%n
```

- 每个category 配置示例：

```
quarkus.log.handler.console.my-console-handler.format=%d{yyyy-MM-dd HH:mm:ss}
[com.example] %s%e%n

quarkus.log.category."com.example".handlers=my-console-handler
quarkus.log.category."com.example".use-parent-handlers=false
```

有关其配置的详情，请参阅 控制台日志记录配置 参考。

1.6.2. 文件日志处理程序

要将事件记录到应用主机上的文件中，请使用 Quarkus 文件日志处理程序。文件日志处理程序默认为禁用，因此您必须首先启用它。

Quarkus 文件日志处理程序支持日志文件轮转。

日志文件轮转通过维护指定数量的备份日志文件来确保一段时间内有效的日志文件管理，同时保持主日志文件最新和可管理。

- 全局配置示例：

```
quarkus.log.file.enable=true
quarkus.log.file.path=application.log
quarkus.log.file.format=%d{yyyy-MM-dd HH:mm:ss} %-5p [%c] (%t) %s%e%n
```

- 每个category 配置示例：

```
quarkus.log.handler.file.my-file-handler.enable=true
quarkus.log.handler.file.my-file-handler.path=application.log
quarkus.log.handler.file.my-file-handler.format=%d{yyyy-MM-dd HH:mm:ss}
[com.example] %s%e%n

quarkus.log.category."com.example".handlers=my-file-handler
quarkus.log.category."com.example".use-parent-handlers=false
```

有关其配置的详情，请查看 文件日志记录配置 参考。

1.6.3. syslog 日志处理程序

Quarkus 中的 syslog 处理程序遵循 Syslog 协议，该协议用于在类似 UNIX 的系统上发送日志消息。它使用 RFC 5424 中定义的协议。

默认情况下禁用 syslog 处理程序。启用后，它会将所有日志事件发送到 syslog 服务器，通常是应用程序的本地 syslog 服务器。

- 全局配置示例：

```
quarkus.log.syslog.enable=true
quarkus.log.syslog.app-name=my-application
quarkus.log.syslog.format=%d{yyyy-MM-dd HH:mm:ss} %-5p [%c] (%t) %s%e%n
```

- 每个category 配置示例：

```
quarkus.log.handler.syslog.my-syslog-handler.enable=true
quarkus.log.handler.syslog.my-syslog-handler.app-name=my-application
quarkus.log.handler.syslog.my-syslog-handler.format=%d{yyyy-MM-dd HH:mm:ss}
[com.example] %s%e%n

quarkus.log.category."com.example".handlers=my-syslog-handler
quarkus.log.category."com.example".use-parent-handlers=false
```

有关其配置的详情，请查看 Syslog 日志记录配置 参考。

1.7. 在日志处理程序中添加日志过滤器

日志处理程序（如控制台日志处理程序）可以与决定是否应记录日志记录 的过滤器 相关联。

注册日志记录过滤器：

1. 使用 @io.quarkus.logging.LoggingFilter 注解实现 java.util.logging.Filter 的最后一个类，并设置 name 属性：

编写过滤器的示例：

```
package com.example;

import io.quarkus.logging.LoggingFilter;
import java.util.logging.Filter;
import java.util.logging.LogRecord;

@LoggingFilter(name = "my-filter")
public final class TestFilter implements Filter {

    private final String part;

    public TestFilter(@ConfigProperty(name = "my-filter.part") String part) {
        this.part = part;
    }

    @Override
    public boolean isLoggable(LogRecord record) {
        return !record.getMessage().contains(part);
    }
}
```

在本例中，我们排除了包含控制台日志中特定文本的日志记录。要过滤的特定文本不是硬编码的，而是从 my-filter.part 配置属性中读取。

在 application.properties 中配置过滤器的示例：

```
my-filter.part=TEST
```

2. 使用位于 application.properties 中的 过滤器 配置属性，将过滤器附加到对应的处理器中：

```
quarkus.log.console.filter=my-filter
```

1.8. 日志配置示例

以下示例显示了您可以在 Quarkus 中配置日志的一些方法：

控制台 DEBUG 日志记录，除了 Quarkus 日志(INFO)，没有颜色、缩短时间、短的类别前缀

```
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
quarkus.log.console.level=DEBUG
quarkus.console.color=false

quarkus.log.category."io.quarkus".level=INFO
```

注意

如果您在命令行中添加这些属性，请确保 " is escaped.例如，-Dquarkus.log.category.\"io.quarkus\".level=DEBUG。

文件 TRACE 日志记录配置

```
quarkus.log.file.enable=true
# Send output to a trace.log file under the /tmp directory
quarkus.log.file.path=/tmp/trace.log
quarkus.log.file.level=TRACE
quarkus.log.file.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
# Set 2 categories (io.quarkus.smallrye.jwt, io.undertow.request.security) to TRACE level
quarkus.log.min-level=TRACE
quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE
quarkus.log.category."io.undertow.request.security".level=TRACE
```

注意

由于我们不更改根日志记录器，控制台日志将仅包含 INFO 或更高级别的日志。

附加到类别的命名处理程序

```
# Send output to a trace.log file under the /tmp directory
quarkus.log.file.path=/tmp/trace.log
quarkus.log.console.format=%d{HH:mm:ss} %-5p [%c{2.}] (%t) %s%e%n
# Configure a named handler that logs to console
quarkus.log.handler.console."STRUCTURED_LOGGING".format=%e%n
# Configure a named handler that logs to file
quarkus.log.handler.file."STRUCTURED_LOGGING_FILE".enable=true
quarkus.log.handler.file."STRUCTURED_LOGGING_FILE".format=%e%n
# Configure the category and link the two named handlers to it
quarkus.log.category."io.quarkus.category".level=INFO
quarkus.log.category."io.quarkus.category".handlers=STRUCTURED_LOGGING,STRUCTURE
D_LOGGING_FILE
```

附加到根日志记录器的命名处理程序

```
# configure a named file handler that sends the output to 'quarkus.log'
quarkus.log.handler.file.CONSOLE_MIRROR.enable=true
quarkus.log.handler.file.CONSOLE_MIRROR.path=quarkus.log
# attach the handler to the root logger
quarkus.log.handlers=CONSOLE_MIRROR
```

1.9. 集中式日志管理

使用集中位置，高效地收集、存储和分析应用各个组件和实例的日志数据。

要将日志发送到集中工具，如 Graylog、Logstash 或 Fluentd，请参阅 Quarkus 集中式日志 管理指南。

1.10. 为 @QUARKUSTEST配置日志记录

通过将 java.util.logging.manager 系统属性设置为 org.jboss.logmanager.LogManager，为 @QuarkusTest 启用正确的日志记录。

系统属性必须在早期设置才能生效，因此建议在构建系统中进行配置。

在 Maven Surefire 插件配置中设置 java.util.logging.manager 系统属性

```
<build>
  <plugins>
    <plugin>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${surefire-plugin.version}</version>
      <configuration>
        <systemPropertyVariables>
          <java.util.logging.manager>org.jboss.logmanager.LogManager</java.util.logging.manager>
(1)
          <quarkus.log.level>DEBUG</quarkus.log.level> (2)
          <maven.home>${maven.home}</maven.home>
        </systemPropertyVariables>
      </configuration>
    </plugin>
  </plugins>
</build>
```

(1) 确保使用了 org.jboss.logmanager.LogManager。

(2) 为所有日志记录类别启用调试日志记录。

对于 Gradle，请在 build.gradle 文件中添加以下配置：

```
test {
 systemProperty "java.util.logging.manager", "org.jboss.logmanager.LogManager"
}
```

请参阅从 IDE 运行 @QuarkusTest 。

1.11. 使用其他日志记录 API

Quarkus 依赖于 JBoss Logging 库来满足所有日志记录要求。

假设您使用依赖于其他日志记录库的库，如 Apache Commons Logging、Log4j 或 SLF4J。在这种情况下，从依赖项中排除它们，并使用其中一个 JBoss Logging 适配器。

这在构建原生可执行文件时尤为重要，因为在编译原生可执行文件时可能会遇到类似如下的问题：

```
Caused by java.lang.ClassNotFoundException: org.apache.commons.logging.impl.LogFactoryImpl
```

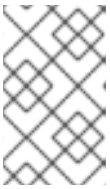
日志记录实施不包括在原生可执行文件中，但您可以使用 JBoss Logging 适配器解决这个问题。

这些适配器可用于流行的开源日志记录组件，如下一章节中所述。

1.11.1. 在应用程序中添加日志适配器

对于不是 jboss-logging 的每个日志记录 API：

1. 添加日志记录适配器库，以确保通过这些 API 记录的消息被路由到 JBoss Log Manager 后端。



注意

对于作为 Quarkus 扩展依赖项的库，这个步骤不需要扩展自动处理它。

- Apache Commons Logging:
  - 使用 Maven：

```
<dependency>
    <groupId>org.jboss.logging</groupId>
    <artifactId>commons-logging-jboss-logging</artifactId>
</dependency>
```

- 使用 Gradle ：

```
implementation("org.jboss.logging:commons-logging-jboss-logging")
```

- Log4j:
  - 使用 Maven：

```
<dependency>
  <groupId>org.jboss.logmanager</groupId>
  <artifactId>log4j-jboss-logmanager</artifactId>
</dependency>
```

  - 使用 Gradle ：

```
implementation("org.jboss.logmanager:log4j-jboss-logmanager")
```

- Log4j 2:
  - 使用 Maven：

```
<dependency>
  <groupId>org.jboss.logmanager</groupId>
  <artifactId>log4j2-jboss-logmanager</artifactId>
</dependency>
```

  - 使用 Gradle ：

```
implementation("org.jboss.logmanager:log4j2-jboss-logmanager")
```

注意

不要包含任何 Log4j 依赖项，因为 log4j2-jboss-logmanager 库包含使用 Log4j 作为日志实施所需的所有操作。

- SLF4J:

- 使用 Maven：

```
<dependency>
    <groupId>org.jboss.slf4j</groupId>
    <artifactId>slf4j-jboss-logmanager</artifactId>
</dependency>
```

- 使用 Gradle ：

```
implementation("org.jboss.slf4j:slf4j-jboss-logmanager")
```

2. 验证添加的库生成的日志是否遵循与其他 Quarkus 日志相同的格式。

1.11.2. 使用 MDC 添加上下文日志信息

Quarkus 覆盖了日志记录映射的诊断上下文(MDC)，以提高其重新活跃内核的兼容性。

1.11.2.1. 添加和读取 MDC 数据

在 MDC 中添加数据并将其提取到日志输出中：

1. 使用 MDC 类来设置数据。
2. 自定义日志格式以使用 %X{mdc-key}。

让我们考虑以下代码：

使用 JBoss Logging 和 io.quarkus.logging.Log的示例

```
package me.sample;

import io.quarkus.logging.Log;
import jakarta.ws.rs.GET;
import jakarta.ws.rs.Path;
import org.jboss.logmanager.MDC;
```

```
import java.util.UUID;

@Path("/hello/jboss")
public class GreetingResourceJbossLogging {

    @GET
    @Path("/test")
    public String greeting() {
        MDC.put("request.id", UUID.randomUUID().toString());
        MDC.put("request.path", "/hello/test");
        Log.info("request received");
        return "hello world!";
    }
}
```

如果您使用以下行配置日志格式：

```
quarkus.log.console.format=%d{HH:mm:ss} %-5p request.id=%X{request.id}
request.path=%X{request.path} [%c{2.}] (%t) %s%n
```

您会收到包含 MDC 数据的消息：

```
08:48:13 INFO request.id=c37a3a36-b7f6-4492-83a1-de41dbc26fe2 request.path=/hello/test
[me.sa.GreetingResourceJbossLogging] (executor-thread-1) request received
```

1.11.2.2. MDC 和支持的日志记录 API

根据您使用的 API，MDC 类略有不同。但是，API 非常相似：

- Log4j 1 - org.apache.log4j.MDC.put (key, value)
- Log4j 2 - org.apache.logging.log4j.ThreadContext.put(key, value)
- SLF4J - org.slf4j.MDC.put(key, value)

1.11.2.3. MDC propagation

在 Quarkus 中，MDC 供应商具有处理被动上下文的特定实现，可确保在被动和异步处理过程中传播

MDC 数据。

因此，您仍可在各种情况下访问 MDC 数据：

- 异步调用后，例如当 REST 客户端返回 Uni 时。
- 在提交至 org.eclipse.microprofile.context.ManagedExecutor 的代码中。
- 在使用 vertx.executeBlocking （） 执行的代码中。

注意

如果适用，MDC 数据存储在 *重复的上下文中*，这是处理单个任务（请求）的隔离上下文。

1.12. 日志记录配置参考

构建时修复的配置属性 - 所有其他配置属性可在运行时覆盖

Configuration 属性	类型	default
quarkus.log.metrics.enabled 如果存在启用和指标扩展，则会发布日志记录指标。 环境变量： QUARKUS_LOG_METRICS_ENABLED	布尔值	false
quarkus.log.min-level 默认最小日志级别。 环境变量： QUARKUS_LOG_MIN_LEVEL	级别	DEBUG
最小日志记录类别	类型	default

quarkus.log.category."categories".min-level 此类别的最低日志级别。默认情况下，所有类别都配置有 **DEBUG** 最小级别。 要在 **DEBUG** 下方获取运行时日志记录，如 **TRACE**，可在构建时调整最小级别。需要在运行时提供正确的日志级别。 例如，若要获取 **TRACE** 日志记录，最小级别需要位于 **TRACE**，运行时日志级别需要匹配： 环境变量：**QUARKUS_LOG_CATEGORY__CATEGORIES__MIN_LEVEL**	InheritableLevel	**inherit**

构建时修复的配置属性 - 所有其他配置属性可在运行时覆盖

Configuration 属性	类型	default
quarkus.log.level root 类别的日志级别，用作所有类别的默认日志级别。 JBoss Logging 支持 Apache 风格的日志级别： • {@link org.jboss.logmanager.Level#FATAL} • {@link org.jboss.logmanager.Level#ERROR} • {@link org.jboss.logmanager.Level#WARN} • {@link org.jboss.logmanager.Level#INFO} • {@link org.jboss.logmanager.Level#DEBUG} • {@link org.jboss.logmanager.Level#TRACE} 另外，它还支持标准 JDK 日志级别。 环境变量：**QUARKUS_LOG_LEVEL**	级别	**INFO**
quarkus.log.handlers 要链接到根类别的额外处理程序的名称。这些处理程序在 consoleHandlers、fileHandlers 或 syslogHandlers 中定义。 环境变量：**QUARKUS_LOG_HANDLERS**	字符串列表	
Console logging	类型	default

quarkus.log.console.enable 如果应该启用控制台日志记录 环境变量： **QUARKUS_LOG_CONSOLE_ENABLE**	布尔值	**true**
quarkus.log.console.stderr 如果控制台日志记录应该进入 **System#err** 而不是 **System#out**。 环境变量： **QUARKUS_LOG_CONSOLE_STDERR**	布尔值	**false**
quarkus.log.console.format 日志格式。请注意，如果扩展出现可控制控制台格式（例如，XML 或 JSON 格式扩展）的扩展，则忽略这个值。 环境变量： **QUARKUS_LOG_CONSOLE_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.console.level 控制台日志级别。 环境变量： **QUARKUS_LOG_CONSOLE_LEVEL**	级别	**ALL**
quarkus.log.console.darken 指定应分类颜色的数量。请注意，如果扩展出现可控制控制台格式（例如，XML 或 JSON 格式扩展）的扩展，则忽略这个值。 环境变量： **QUARKUS_LOG_CONSOLE_DARKEN**	int	**0**
quarkus.log.console.filter 要链接到控制台处理程序的过滤器名称。 环境变量： **QUARKUS_LOG_CONSOLE_FILTER**	string	
quarkus.log.console.async 指明是否异步登录 环境变量： **QUARKUS_LOG_CONSOLE_ASYNC**	布尔值	**false**

quarkus.log.console.async.queue-length 在清除写前使用的队列长度 环境变量： **QUARKUS_LOG_CONSOLE_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.console.async.overflow 确定在队列满时是否阻止发布者（而不是丢弃消息） 环境变量： **QUARKUS_LOG_CONSOLE_ASYNC_OVERFLOW**	**块,discard**	**block**
File logging	**类型**	**default**
quarkus.log.file.enable 应该启用文件日志记录 环境变量： **QUARKUS_LOG_FILE_ENABLE**	布尔值	**false**
quarkus.log.file.format 日志格式 环境变量： **QUARKUS_LOG_FILE_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %h %N[%i] %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.file.level 要写入文件中的日志级别。 环境变量： **QUARKUS_LOG_FILE_LEVEL**	级别	**ALL**
quarkus.log.file.path 将写入日志的文件的名称。 环境变量： **QUARKUS_LOG_FILE_PATH**	File	**quarkus.log**

quarkus.log.file.filter 要链接到文件处理程序的过滤器名称。 环境变量： **QUARKUS_LOG_FILE_FILTER**	string	
quarkus.log.file.encoding 使用的字符编码 环境变量： **QUARKUS_LOG_FILE_ENCODING**	charset	
quarkus.log.file.async 指明是否异步登录 环境变量： **QUARKUS_LOG_FILE_ASYNC**	布尔值	**false**
quarkus.log.file.async.queue-length 在清除写前使用的队列长度 环境变量： **QUARKUS_LOG_FILE_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.file.async.overflow 确定在队列满时是否阻止发布者（而不是丢弃消息） 环境变量： **QUARKUS_LOG_FILE_ASYNC_OVERFLOW**	**块,discard**	**block**
quarkus.log.file.rotation.max-file-size 执行轮转的最大日志文件大小。 环境变量： **QUARKUS_LOG_FILE_ROTATION_MAX_FILE_SIZE**	MemorySize	**10M**
quarkus.log.file.rotation.max-backup-index 要保留的最大备份数量。 环境变量： **QUARKUS_LOG_FILE_ROTATION_MAX_BACKUP_INDEX**	int	**5**
quarkus.log.file.rotation.file-suffix 文件处理程序轮转文件后缀。使用时，该文件将根据其后缀进行轮转。 fileSuffix 示例：.yyyy-MM-dd 注： 如果后缀以 .zip 或 .gz 结尾，则轮转文件也会被压缩。 环境变量： **QUARKUS_LOG_FILE_ROTATION_FILE_SUFFIX**	string	

quarkus.log.file.rotation.rotate-on-boot 指明是否在服务器初始化中轮转日志文件。 您需要设置 **max-file-size** 或配置 **file-suffix** 才能正常工作。 环境变量： **QUARKUS_LOG_FILE_ROTATION_ROTATE_ON_BOOT**	布尔值	**true**
Syslog 日志记录	**类型**	**default**
quarkus.log.syslog.enable 如果应该启用 syslog 日志记录 环境变量： **QUARKUS_LOG_SYSLOG_ENABLE**	布尔值	**false**
quarkus.log.syslog.endpoint Syslog 服务器的 IP 地址和端口 环境变量： **QUARKUS_LOG_SYSLOG_ENDPOINT**	host:port	**localhost:514**
quarkus.log.syslog.app-name 以 RFC5424 格式格式化消息时使用的应用程序名称 环境变量： **QUARKUS_LOG_SYSLOG_APP_NAME**	string	
quarkus.log.syslog.hostname 从发送消息的主机名称 环境变量： **QUARKUS_LOG_SYSLOG_HOSTNAME**	string	

quarkus.log.syslog.facility 设置计算 RFC-5424 和 RFC-3164 定义时消息优先级的工具 环境变量： **QUARKUS_LOG_SYSLOG_FACILITY**	**内核,用户级,mail-system,system-daemons,安全,syslogd,line-printer,network-news,uucp,clock-daemon,security2,ftp-daemon,ntp,log-audit,log-alert,clock-daemon2,local-use-0,local-use-1,local-use-2,local-use-3,local-use-4,local-use-5,local-use-6,local-use-7**	**user-level**

quarkus.log.syslog.syslog-type 设置 **SyslogType syslog 类型** 此处理程序应该用来格式化发送的消息 环境变量： **QUARKUS_LOG_SYSLOG_SYSLOG_TYPE**	**rfc5424, rfc3164**	**rfc5424**
quarkus.log.syslog.protocol 设置用于连接 Syslog 服务器的协议 环境变量： **QUARKUS_LOG_SYSLOG_PROTOCOL**	**tcp,udp,ssl-tcp**	**tcp**
quarkus.log.syslog.use-counting-framing 如果启用，将发送的消息前缀为消息的大小 环境变量： **QUARKUS_LOG_SYSLOG_USE_COUNTING_FRAMING**	布尔值	**false**
quarkus.log.syslog.truncate 设置为 **true**，如果消息超过最大长度，则截断消息 环境变量： **QUARKUS_LOG_SYSLOG_TRUNCATE**	布尔值	**true**
quarkus.log.syslog.block-on-reconnect 尝试重新连接 **org.jboss.logmanager.handlers.SyslogHandler.Protocol#TCP TCP** 或 **org.jboss.logmanager.handlers.SyslogHandler.Protocol#SSL_TCP SSL TCP** 协议时启用或禁用阻塞 环境变量： **QUARKUS_LOG_SYSLOG_BLOCK_ON_RECONNECT**	布尔值	**false**
quarkus.log.syslog.format 日志消息格式 环境变量： **QUARKUS_LOG_SYSLOG_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.syslog.level 指定 Syslog 日志记录器将记录哪些消息级别的日志级别 环境变量： **QUARKUS_LOG_SYSLOG_LEVEL**	级别	**ALL**

quarkus.log.syslog.filter 要链接到文件处理程序的过滤器名称。 环境变量： **QUARKUS_LOG_SYSLOG_FILTER**	string	
quarkus.log.syslog.max-length 允许发送消息的最大长度（以字节为单位）。长度包括标头和消息。 如果没有设置，则当 **sys-log-type** 是 **rfc5424**（默认值）和 **1024** 时，当 **sys-log-type** 为 **rfc3164**时，默认值为 **2048** 环境变量： **QUARKUS_LOG_SYSLOG_MAX_LENGTH**	MemorySize	
quarkus.log.syslog.async 指明是否异步登录 环境变量： **QUARKUS_LOG_SYSLOG_ASYNC**	布尔值	**false**
quarkus.log.syslog.async.queue-length 在清除写前使用的队列长度 环境变量： **QUARKUS_LOG_SYSLOG_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.syslog.async.overflow 确定在队列满时是否阻止发布者（而不是丢弃消息） 环境变量： **QUARKUS_LOG_SYSLOG_ASYNC_OVERFLOW**	**块,discard**	**block**
Logging 类别	**类型**	**default**
quarkus.log.category."categories".level 此类别的日志级别。 请注意，若要在 **INFO** 下方获取日志级别，还需要调整最低级别的构建时间配置选项。 环境变量： **QUARKUS_LOG_CATEGORY__CATEGORIES__LEVEL**	InheritableLevel	**inherit**
quarkus.log.category."categories".handlers 要链接到此类别的处理程序的名称。 环境变量： **QUARKUS_LOG_CATEGORY__CATEGORIES__HANDLERS**	字符串列表	

quarkus.log.category."categories".use-parent-handlers 指定此日志记录器是否应该将其输出发送到其父 Logger 环境变量： **QUARKUS_LOG_CATEGORY__CATEGORIES__USE_PARENT_HANDLERS**	布尔值	**true**
Console handlers	类型	**default**
quarkus.log.handler.console."console-handlers".enable 如果应该启用控制台日志记录 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ENABLE**	布尔值	**true**
quarkus.log.handler.console."console-handlers".stderr 如果控制台日志记录应该进入 **System#err** 而不是 **System#out**。 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__STDERR**	布尔值	**false**
quarkus.log.handler.console."console-handlers".format 日志格式。请注意，如果扩展出现可控制控制台格式（例如，XML 或 JSON 格式扩展）的扩展，则忽略这个值。 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.handler.console."console-handlers".level 控制台日志级别。 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__LEVEL**	级别	**ALL**

quarkus.log.handler.console."console-handlers".darken 指定应分类颜色的数量。请注意，如果扩展出现可控制控制台格式（例如，XML 或 JSON 格式扩展）的扩展，则忽略这个值。 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__DARKEN**	int	**0**
quarkus.log.handler.console."console-handlers".filter 要链接到控制台处理程序的过滤器名称。 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__FILTER**	string	
quarkus.log.handler.console."console-handlers".async 指明是否异步登录 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC**	布尔值	**false**
quarkus.log.handler.console."console-handlers".async.queue-length 在清除写前使用的队列长度 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.console."console-handlers".async.overflow 确定在队列满时是否阻止发布者（而不是丢弃消息） 环境变量： **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC_OVERFLOW**	块,discard	**block**
File handlers	类型	default
quarkus.log.handler.file."file-handlers".enable 应该启用文件日志记录 环境变量：**QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ENABLE**	布尔值	**false**

quarkus.log.handler.file."file-handlers".format 日志格式 环境变量：**QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %h %N[%i] %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.handler.file."file-handlers".level 要写入文件中的日志级别。 环境变量：**QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__LEVEL**	级别	**ALL**
quarkus.log.handler.file."file-handlers".path 将写入日志的文件的名称。 环境变量：**QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__PATH**	File	**quarkus.log**
quarkus.log.handler.file."file-handlers".filter 要链接到文件处理程序的过滤器名称。 环境变量：**QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__FILTER**	string	
quarkus.log.handler.file."file-handlers".encoding 使用的字符编码 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ENCODING**	charset	
quarkus.log.handler.file."file-handlers".async 指明是否异步登录 环境变量：**QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC**	布尔值	**false**

quarkus.log.handler.file."file-handlers".async.queue-length 在清除写前使用的队列长度 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.file."file-handlers".async.overflow 确定在队列满时是否阻止发布者（而不是丢弃消息） 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC_OVERFLOW**	**块,discard**	**block**
quarkus.log.handler.file."file-handlers".rotation.max-file-size 执行轮转的最大日志文件大小。 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_MAX_FILE_SIZE**	MemorySize	**10M**
quarkus.log.handler.file."file-handlers".rotation.max-backup-index 要保留的最大备份数量。 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_MAX_BACKUP_INDEX**	int	**5**
quarkus.log.handler.file."file-handlers".rotation.file-suffix 文件处理程序轮转文件后缀。使用时，该文件将根据其后缀进行轮转。 fileSuffix 示例：.yyyy-MM-dd 注： 如果后缀以 .zip 或 .gz 结尾，则轮转文件也会被压缩。 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_FILE_SUFFIX**	string	
quarkus.log.handler.file."file-handlers".rotation.rotate-on-boot 指明是否在服务器初始化中轮转日志文件。 您需要设置 **max-file-size** 或配置 **file-suffix** 才能正常工作。 环境变量： **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_ROTATE_ON_BOOT**	布尔值	**true**

Syslog handlers	类型	default
quarkus.log.handler.syslog."syslog-handlers".enable 如果应该启用 syslog 日志记录 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ENABLE**	布尔值	**false**
quarkus.log.handler.syslog."syslog-handlers".endpoint Syslog 服务器的 IP 地址和端口 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ENDPOINT**	host:port	**localhost:514**
quarkus.log.handler.syslog."syslog-handlers".app-name 以 RFC5424 格式格式化消息时使用的应用程序名称 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__APP_NAME**	string	
quarkus.log.handler.syslog."syslog-handlers".hostname 从发送消息的主机名称 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__HOSTNAME**	string	

quarkus.log.handler.syslog."syslog-handlers".facility 设置计算 RFC-5424 和 RFC-3164 定义时消息优先级的工具 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FACILITY**	**内核,用户级,mail-system,system-daemons,安全,syslogd,line-printer,network-news,uucp,clock-daemon,security2,ftp-daemon,ntp,log-audit,log-alert,clock-daemon2,local-use-0,local-use-1,local-use-2,local-use-3,local-use-4,local-use-5,local-use-6,local-use-7**	**user-level**

quarkus.log.handler.syslog."syslog-handlers".syslog-type 设置 **SyslogType syslog 类型** 此处理程序应该用来格式化发送的消息 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__SYSLOG_TYPE**	**rfc5424, rfc3164**	**rfc5424**
quarkus.log.handler.syslog."syslog-handlers".protocol 设置用于连接 Syslog 服务器的协议 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS_PROTOCOL**	**tcp,udp,ssl-tcp**	**tcp**
quarkus.log.handler.syslog."syslog-handlers".use-counting-framing 如果启用，将发送的消息前缀为消息的大小 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__USE_COUNTING_FRAMING**	布尔值	**false**
quarkus.log.handler.syslog."syslog-handlers".truncate 设置为 **true**，如果消息超过最大长度，则截断消息 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__TRUNCATE**	布尔值	**true**
quarkus.log.handler.syslog."syslog-handlers".block-on-reconnect 尝试重新连接 **org.jboss.logmanager.handlers.SyslogHandler.Protocol#TCP TCP** 或 **org.jboss.logmanager.handlers.SyslogHandler.Protocol#SSL_TCP SSL TCP** 协议时启用或禁用阻塞 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__BLOCK_ON_RECONNECT**	布尔值	**false**

quarkus.log.handler.syslog."syslog-handlers".format 日志消息格式 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.handler.syslog."syslog-handlers".level 指定 Syslog 日志记录器将记录哪些消息级别的日志级别 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__LEVEL**	级别	**ALL**
quarkus.log.handler.syslog."syslog-handlers".filter 要链接到文件处理程序的过滤器名称。 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FILTER**	string	
quarkus.log.handler.syslog."syslog-handlers".max-length 允许发送消息的最大长度（以字节为单位）。长度包括标头和消息。 如果没有设置，则当 **sys-log-type** 是 **rfc5424**（默认值）和 **1024** 时，当 **sys-log-type** 为 **rfc3164**时，默认值为 **2048** 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__MAX_LENGTH**	MemorySize	
quarkus.log.handler.syslog."syslog-handlers".async 指明是否异步登录 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC**	布尔值	**false**
quarkus.log.handler.syslog."syslog-handlers".async.queue-length 在清除写前使用的队列长度 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**

quarkus.log.handler.syslog."syslog-handlers".async.overflow 确定在队列满时是否阻止发布者（而不是丢弃消息） 环境变量： **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC_OVERFLOW**	块,discard	**block**
Log cleanup filters - 内部使用	类型	default
quarkus.log.filter."filters".if-starts-with 要匹配的消息前缀 环境变量：**QUARKUS_LOG_FILTER__FILTERS__IF_STARTS_WITH**	字符串列表	**inherit**
quarkus.log.filter."filters".target-level 过滤消息的新日志级别。默认为 DEBUG。 环境变量：**QUARKUS_LOG_FILTER__FILTERS__TARGET_LEVEL**	级别	**DEBUG**

关于 MEMORYSIZE 格式

一个大小配置选项 recognises 字符串（显示为正则表达式）：[0-9]+ [KkMmGgTtPpEeZzYy]?.如果未指定后缀，则假定为字节。