Red Hat build of Quarkus 3.2

ロギングの設定

Last Updated: 2024-05-10

法律上の通知

概要

Red Hat build of Quarkus でのロギング API の使用、ロギング出力の設定、統合出力のためのロギングアダプターの使用についてお読みください。

目次

多様性を受け入れるオープンソースの強化 **3**

第1章 ロギングの設定 **4**

1.1. JBOSS LOGGING を使用するアプリケーションロギング 4
1.2. アプリケーションロガーの取得 5
1.3. ログレベルの使用 7
1.4. ログレベル、カテゴリー、フォーマットの設定 8
1.5. ロギングフォーマット 11
1.6. ログハンドラー 19
1.7. ログハンドラーにロギングフィルターを追加する 20
1.8. ロギング設定の例 21
1.9. ログの一元管理 22
1.10. @QUARKUSTEST のロギング設定 22
1.11. 他のロギング API を使用する 23
1.12. ロギング設定リファレンス 26

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みは膨大な作業を要するため、今後の複数のリリースで段階的に用語の置き換えを実施して参ります。詳細は、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 ロギングの設定

Quarkus でのロギング API の使用、ロギング出力の設定、統合出力のためのロギングアダプターの使用についてお読みください。

Quarkus は、アプリケーションおよびフレームワークログの公開に、JBoss Log Manager ロギングバックエンドを使用します。Quarkus は、JBoss Log Manager とシームレスに統合された、JBoss Logging API およびその他の複数のロギング API をサポートします。以下の API のいずれかを使用できます。

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

1.1. JBOSS LOGGING を使用するアプリケーションロギング

JBoss Logging API を使用する場合、Quarkus が自動的に提供するため、アプリケーションに追加の依存関係は必要ありません。

JBoss Logging API を使用してメッセージをログに記録する例:

```
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";
    }
}
```

注記

JBoss Logging は、ログメッセージを JBoss Log Manager に直接ルーティングしますが、別のロギング API に依存するライブラリーがある可能性もあります。その場合、ロギングアダプター を使用して、ログメッセージも JBoss Log Manager にルーティングされるようにする必要があります。

1.2. アプリケーションロガーの取得

以下は、最も一般的な Quarkus でのアプリケーションロガー取得方法です。

- ロガーフィールドを宣言する
- ロギングを簡素化する
- 設定されたロガーを注入する

1.2.1. ロガーフィールドを宣言する

この従来のアプローチでは、特定の API を使用してロガーインスタンスを取得し、それをクラスの静的フィールドに格納し、このインスタンスに対してロギング操作を呼び出します。

サポートされているロギング API であれば、同じフローを適用できます。

JBoss Logging API を使用してロガーインスタンスを静的フィールドに保存する例:

```
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) ロガーフィールドを定義します。

(2) **log** オブジェクト上で目的のロギングメソッドを呼び出します。

1.2.2. ロギングを簡素化する

Quarkus は、**io.quarkus.logging.Log** を使用するクラスにロガーフィールドを自動的に追加することで、ロギングを簡素化します。これにより、反復的なボイラープレートコードが不要になり、ロギング設定の利便性が向上します。

静的メソッド呼び出しを使用して簡素化されたロギングの例:

```
package com.example;

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

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

(1) **io.quarkus.logging.Log** クラスには、JBoss Logging と同じメソッドが含まれます。ただし、それが static である点は異なります。

れか **static** である点は異なります。

2 クラスはロガーフィールドを宣言しないことに注意してください。これは、アプリケーションのビルド中に、**Log** API を使用する各クラスに **private static final org.jboss.logging.Logger** フィールドが自動的に作成されるためです。**Log** メソッドを呼び出すクラスの完全修飾名は、ロガー名として使用されます。この例のロガー名は **com.example.MyService** です。

3 最後に、**Log** メソッドへの呼び出しはすべて、アプリケーションのビルド中にロガーフィールドに対する通常の JBoss Logging 呼び出しに書き換えられます。

警告

Log API は、外部の依存関係ではなく、アプリケーションクラスでのみ使用してください。ビルド時に Quarkus が処理しない **Log** メソッド呼び出しは、例外を出力します。

1.2.3. 設定されたロガーを注入する

@Inject アノテーションを使用して、設定済みの **org.jboss.logging.Logger** ロガーインスタンスを注入する方法は、アプリケーションアプリケーションロガーを追加する場合の代替手段ですが、これを適用できるのは CDI Bean のみです。

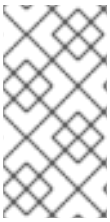
ロガーの名前は、**@Inject Logger log** を使用すると注入するクラスにちなんだ名前、**@Inject @LoggerName("...") Logger log** を使用すると特定の名前になります。注入が完了すると、**log** オブジェクトを使用してロギングメソッドを呼び出せます。

2 種類のロガーインジェクションの例:

```
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 宣言するクラスの FQCN は、ロガー名として使用されます。たとえば、**org.jboss.logging.Logger.getLogger(SimpleBean.class)** が使用されます。

2 この場合、ロガー名として **foo** という名前が使用されます。たとえば、**org.jboss.logging.Logger.getLogger("foo")** が使用されます。



注記

ロガーインスタンスは内部的にキャッシュされます。したがって、たとえば **@RequestScoped** Bean にロガーが注入されると、挿入されると、ロガーのインスタンス化に関連してパフォーマンスが低下することを回避するために、そのロガーはすべての Bean インスタンスで共有されます。

1.3. ログレベルの使用

Quarkus ではさまざまなログレベルが用意されているため、開発者はそれらを使用して、ログに記録する情報の量をイベントの重大度に基づき制御できます。

表1.1 Quarkus で使用されるログレベル

OFF	ロギングをオフにするために設定で使用する特別なレベル。
FATAL	重大なサービス障害、またはあらゆるタイプのリクエストにまったく対応できない場合。
ERROR	リクエストの大幅な中断、リクエストに対応できない場合。
WARN	重大ではないサービスエラー、またはすぐに是正する必要がない可能性のある問題。
INFO	サービスライフサイクルイベント、または非常に低頻度の重要な関連情報。
DEBUG	ライフサイクルまたはリクエストにバインドされていないイベントに関する追加情報を伝えるメッセージ。デバッグに役立ちます。
TRACE	非常に頻繁に発生する可能性のある追加の要求ごとのデバッグ情報を伝えるメッセージ。
ALL	カスタムレベルを含むすべてのメッセージのログを有効にする場合に設定で使用する特別なレベル。

java.util.logging を使用するアプリケーションおよびライブラリーに対しては、以下のレベルも設定できます。

SEVERE	**ERROR** と同じ。
WARNING	**WARN** と同じ。
CONFIG	サービス設定情報。
FINE	**DEBUG** と同じ。
FINER	**TRACE** と同じ。
FINEST	デバッグ出力が **TRACE** よりも大きくなり、頻度も高い場合があります。

表1.2 レベル間のマッピング

数値で示したレベル値	標準レベル名	該当する java.util.logging (JUL) レベル名
1100	FATAL	該当なし
1000	ERROR	SEVERE
900	WARN	WARNING
800	INFO	INFO
700	該当なし	CONFIG
500	DEBUG	FINE
400	TRACE	FINER
300	該当なし	FINEST

1.4. ログレベル、カテゴリー、フォーマットの設定

JBoss Logging は Quarkus に組み込まれており、すべての サポートされるロギング API で 統合設定 を使用できます。

application.properties ファイルで、ランタイムロギングを設定します。

デフォルトのログレベルを INFO ロギングに設定し、Hibernate DEBUG ログを含める例:

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

ログレベルを **DEBUG** 未満に設定する場合は、最小ログレベルも調整する必要があります。これは、**quarkus.log.min-level** 設定プロパティーを使用してグローバルに、またはカテゴリーごとに設定できます。

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

これにより、Quarkus がサポートコードを生成する必要があるフロアレベルが設定されます。Quarkus が実用的ではないレベルのログを省略して最適化できるように、最小ログレベルはビルド時に設定する必要があります。

ネイティブ実行の例:

INFO を最小ログレベルとして設定すると、それより低い **isTraceEnabled** などのレベルのチェックが **false** に設定されます。これにより、**if(logger.isDebug()) callMethod();** のようなコードが特定され、"dead" としてマークされて決して実行されなくなります。

警告

該当するプロパティーをコマンドラインで追加する場合は、" の文字が正しくエスケープされていることを確認してください。

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

可能性のあるプロパティーは、すべて ロギング設定リファレンス セクションにリストされています。

1.4.1. ロギングカテゴリー

ロギングはカテゴリーごとに設定され、各カテゴリーは個別に設定されます。具体的なサブカテゴリー設定がなければ、カテゴリーの設定はすべてのサブカテゴリーに再帰的に適用されます。

すべてのロギングカテゴリーの親は、"ルートカテゴリー" と呼ばれます。 最終的な親となるこのカテゴリーには、他の全カテゴリーにグローバルに適用される設定が含まれる場合があります。これには、グローバルに設定されたハンドラーおよびフォーマッターが含まれます。

例1.1 すべてのカテゴリーに適用されるグローバル設定の例:

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

この例では、ルートカテゴリーは **console** および **mylog** の 2 つのハンドラーを使用するよう設定されています。

例1.2 カテゴリーごとの設定の例:

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

この例は、**org.apache.kafka.clients** および **org.apache.kafka.common.utils** のカテゴリーで、最小ログレベルを設定する方法を示しています。

詳細は、ロギング設定リファレンス を参照してください。

特定のカテゴリーに設定を追加する場合は、**quarkus.log.handler.[console|file|syslog].<your-handler-name>.*** などの名前付きハンドラーを作成し、**quarkus.log.category.<my-category>.handlers** を使用してそのカテゴリーに設定します。

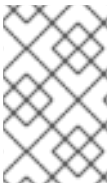
ユースケースの例としては、ファイルに保存されるログメッセージに、他のハンドラーで使用されるフォーマットとは異なるタイムスタンプフォーマットを使用する場合があります。

詳細なデモは、b名前付きハンドラーをカテゴリーに割り当てる の出力例を参照してください。

プロパティー名	デフォルト	説明
quarkus.log.category."<category-name>".level	**INFO** [a]	**<category-name>** という名前のカテゴリーの設定に使用するレベル。引用符が必要です。
quarkus.log.category."<category-name>".min-level	**DEBUG**	**<category-name>** という名前のカテゴリーの設定に使用する最小ログレベル。引用符が必要です。
quarkus.log.category."<category-name>".use-parent-handlers	**true**	このロガーが出力を親ロガーに送信するかどうかを指定します。
quarkus.log.category."<category-name>".handlers=[<handler>]	**empty** [b]	特定のカテゴリーに割り当てるハンドラーの名前。

[a] 一部のエクステンションでは、デフォルトでログノイズを減らすために、特定のカテゴリーにカスタマイズされたデフォルトログレベルを定義する場合があります。設定でログレベルを設定すると、エクステンションで定義されたログレベルがオーバーライドされます。

[b] デフォルトで、設定されたカテゴリーのハンドラーは、ルートロガーに割り当てられたハンドラーと同じになります。



注記

. 記号は、設定プロパティーの特定の部分を分離します。プロパティー名に含まれる引用符は、カテゴリー仕様をそのまま維持するために必要なエスケープとして使用されます (例: **quarkus.log.category."io.quarkus.smallrye.jwt".level=TRACE**)。

1.4.2. ルートロガー設定

ルートロガーカテゴリーは個別に処理され、次のプロパティーを使用して設定されます。

プロパティー名	デフォルト	説明
quarkus.log.level	**INFO**	すべてのログカテゴリーのデフォルトログレベル。
quarkus.log.min-level	**DEBUG**	すべてのログカテゴリーのデフォルト最小ログレベル。

- 指定されたロガーカテゴリーにレベルが設定されていない場合は、親カテゴリーが確認されます。
- カテゴリーやその親カテゴリーのいずれにも設定が指定されていない場合は、ルートロガー設定が使用されます。

注記

通常、ルートロガーのハンドラーは **quarkus.log.console**、**quarkus.log.file**、**quarkus.log.syslog** を介して直接設定されますが、**quarkus.log.handlers** プロパティーを使用して追加の名前付きハンドラーを割り当てることも可能です。

1.5. ロギングフォーマット

Quarkus は、デフォルトで人間が判読できるテキストログを生成するパターンベースのロギングフォーマッターを使用しますが、専用プロパティーを使用して各ログハンドラーのフォーマットを設定することもできます。

コンソールハンドラーの場合、プロパティーは **quarkus.log.console.format** です。

ロギングフォーマットの文字列では、次のシンボルを使用できます。

記号	概要	説明
%%	**%**	単純な **%** 文字をレンダリングします。
%c	カテゴリー	カテゴリー名をレンダリングします。
%C	ソースクラス	ソースクラス名をレンダリングします。 footnote:calc:calc[呼び出し元情報を調べるフォーマットシーケンスはパフォーマンスに影響を及ぼす可能性があります。]
%d{xxx}	日付	指定された日付フォーマットの文字列で日付をレンダリングします。**java.text.SimpleDateFormat** で定義された構文を使用します。
%e	例外	発生した例外がある場合はそれをレンダリングします。
%F	ソースファイル	ソースファイル名をレンダリングします。footnote:calc[]
%h	ホスト名	システムの単純なホスト名をレンダリングします。
%H	修飾ホスト名	システムの完全修飾ホスト名を表示します。オペレーティングシステムの設定によっては、単純なホスト名と同じになる場合があります。
%i	プロセス ID	現在のプロセス PID をレンダリングします。
%l	ソースの場所	ソースファイル名、行番号、クラス名、メソッド名を含むソースの場所情報をレンダリングします。footnote:calc[]
%L	ソース行	ソース行番号をレンダリングします。footnote:calc[]

記号	概要	説明
%m	フルメッセージ	ログメッセージと例外 (存在する場合) をレンダリングします。
%M	ソースメソッド	ソースメソッド名をレンダリングします。footnote:calc[]
%n	改行	プラットフォーム固有の行区切り文字列をレンダリングします。
%N	プロセス名	現在のプロセスの名前をレンダリングします。
%p	レベル	メッセージのログレベルをレンダリングします。
%r	相対時間	アプリケーションログの開始からの相対時間 (ミリ秒単位) をレンダリングします。
%s	単純なメッセージ	例外トレースなしで、ログメッセージのみをレンダリングします。
%t	スレッド名	スレッド名をレンダリングします。
%t{id}	スレッド ID	スレッド ID をレンダリングします。
%z{<zone name>}	タイムゾーン	出力のタイムゾーンを **<zone name>** に設定します。
%X{<MDC property name>}	マッピングされた診断コンテキスト値	マッピングされた診断コンテキストから値をレンダリングします。
%X	マッピングされた診断コンテキスト値 (複数)	マッピングされた診断コンテキストから、すべての値を {property.key=property.value} フォーマットでレンダリングします。
%x	ネスト化された診断コンテキスト値	ネストされた診断コンテキストから、すべての値 を {value1.value2} フォーマットでレンダリングします。

1.5.1. 代替のコンソールロギングフォーマット

コンソールログフォーマットは、たとえば後で分析するためにログ情報を処理および保存するサービスが Quarkus アプリケーションのコンソール出力をキャプチャーする場合などに変更できます。

1.5.1.1. JSON ロギングフォーマット

quarkus-logging-json エクステンションを使用して、JSON ロギングフォーマットとその関連設定のサポートを追加できます。

次の抜粋が示すとおり、このエクステンションをビルドファイルに追加します。

- Maven の使用:

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

- Gradle を使用する場合:

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

デフォルトでは、このエクステンションが存在するとコンソール設定の出力フォーマット設定が置き換えられ、フォーマット文字列と色設定 (存在する場合) は無視されます。非同期ログやログレベルの制御など、他のコンソール設定項目は引き続き適用されます。

開発モードでは人間が判読できる (非構造化) ログを使用し、実稼働モードでは JSON ログ (構造化) を使用することが合理的なこともあります。その場合は、次の設定に示すように、さまざまなプロファイルを使用できます。

開発モードおよびテストモードで application.properties の JSON ロギングを無効にする

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

1.5.1.1.1. 設定

サポートされるプロパティーを使用して JSON ロギングエクステンションを設定し、動作をカスタマイズします。

ビルド時に固定された設定プロパティー: その他の設定プロパティーはすべてランタイム時にオーバーライド可能

コンソールのロギング	型	デフォルト
quarkus.log.console.json JSON コンソールフォーマットエクステンションを有効にするかどうかを決定します。これにより、"通常の" コンソールフォーマットが無効になります。 環境変数: **QUARKUS_LOG_CONSOLE_JSON**	boolean	**true**
quarkus.log.console.json.pretty-print JSON レコードの "プリティープリント" を有効にします。一部の JSON パーサーは、プリティープリント出力を読み取れないことに注意してください。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_PRETTY_PRINT**	boolean	**false**

quarkus.log.console.json.date-format 使用する日付フォーマット。"default" は特別な文字列で、デフォルトフォーマットを使用する必要があることを示します。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_DATE_FORMAT**	string	**default**
quarkus.log.console.json.record-delimiter 使用する特別なレコード終了区切り文字。デフォルトでは、改行が使用されます。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_RECORD_DELIMITER**	string	
quarkus.log.console.json.zone-id 使用するゾーン ID。"default" は特別な文字列で、デフォルトのゾーンを使用する必要があることを示します。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_ZONE_ID**	string	**default**
quarkus.log.console.json.exception-output-type 指定する例外出力型。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_EXCEPTION_OUTPUT_TYPE**	**detailed**, **formatted**, **detailed-and-formatted**	**detailed**
quarkus.log.console.json.print-details ログでの詳細出力を有効にします。 値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。詳細には、ソースクラス名、ソースファイル名、ソースメソッド名、およびソース行番号が含まれます。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_PRINT_DETAILS**	boolean	**false**
quarkus.log.console.json.key-overrides カスタム値でキーをオーバーライドします。この値を省略すると、キーはオーバーライドされません。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_KEY_OVERRIDES**	string	
quarkus.log.console.json.excluded-keys JSON 出力から除外するキー。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_EXCLUDED_KEYS**	文字列のリスト	

quarkus.log.console.json.additional-field."field-name".value 追加フィールドの値。 環境変数: **QUARKUS_LOG_CONSOLE_JSON_ADDITIONAL_FIELD__FIELD_NAME__VALUE**	string	required
quarkus.log.console.json.additional-field."field-name".type 追加フィールドの型指定。サポートされている型: string、int、long (指定されていない場合は string がデフォルト) 環境変数: **QUARKUS_LOG_CONSOLE_JSON_ADDITIONAL_FIELD__FIELD_NAME__TYPE**	**string**, **int**, **long**	**string**
ファイルのロギング	**型**	**デフォルト**
quarkus.log.file.json JSON コンソールフォーマットエクステンションを有効にするかどうかを決定します。これにより、"通常の" コンソールフォーマットが無効になります。 環境変数: **QUARKUS_LOG_FILE_JSON**	boolean	**true**
quarkus.log.file.json.pretty-print JSON レコードの "プリティープリント" を有効にします。一部の JSON パーサーは、プリティープリント出力を読み取れないことに注意してください。 環境変数: **QUARKUS_LOG_FILE_JSON_PRETTY_PRINT**	boolean	**false**
quarkus.log.file.json.date-format 使用する日付フォーマット。"default" は特別な文字列で、デフォルトフォーマットを使用する必要があることを示します。 環境変数: **QUARKUS_LOG_FILE_JSON_DATE_FORMAT**	string	**default**
quarkus.log.file.json.record-delimiter 使用する特別なレコード終了区切り文字。デフォルトでは、改行が使用されます。 環境変数: **QUARKUS_LOG_FILE_JSON_RECORD_DELIMITER**	string	

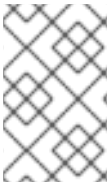
quarkus.log.file.json.zone-id 使用するゾーン ID。"default" は特別な文字列で、デフォルトのゾーンを使用する必要があることを示します。 環境変数: **QUARKUS_LOG_FILE_JSON_ZONE_ID**	string	**default**
quarkus.log.file.json.exception-output-type 指定する例外出力型。 環境変数: **QUARKUS_LOG_FILE_JSON_EXCEPTION_OUTPUT_TYPE**	**detailed**, **formatted**, **detailed-and-formatted**	**detailed**
quarkus.log.file.json.print-details ログでの詳細出力を有効にします。 値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。詳細には、ソースクラス名、ソースファイル名、ソースメソッド名、およびソース行番号が含まれます。 環境変数: **QUARKUS_LOG_FILE_JSON_PRINT_DETAILS**	boolean	**false**
quarkus.log.file.json.key-overrides カスタム値でキーをオーバーライドします。この値を省略すると、キーはオーバーライドされません。 環境変数: **QUARKUS_LOG_FILE_JSON_KEY_OVERRIDES**	string	
quarkus.log.file.json.excluded-keys JSON 出力から除外するキー。 環境変数: **QUARKUS_LOG_FILE_JSON_EXCLUDED_KEYS**	文字列のリスト	
quarkus.log.file.json.additional-field."field-name".value 追加フィールドの値。 環境変数: **QUARKUS_LOG_FILE_JSON_ADDITIONAL_FIELD__FIELD_NAME__VALUE**	string	required

quarkus.log.file.json.additional-field."field-name".type 追加フィールドの型指定。サポートされている型: string、int、long (指定されていない場合は string がデフォルト) 環境変数: **QUARKUS_LOG_FILE_JSON_ADDITIONAL_FIELD__FIELD_NAME__TYPE**	**string**, **int**, **long**	**string**
Syslog ロギング	**型**	**デフォルト**
quarkus.log.syslog.json JSON コンソールフォーマットエクステンションを有効にするかどうかを決定します。これにより、"通常の" コンソールフォーマットが無効になります。 環境変数: **QUARKUS_LOG_SYSLOG_JSON**	boolean	**true**
quarkus.log.syslog.json.pretty-print JSON レコードの "プリティープリント" を有効にします。一部の JSON パーサーは、プリティープリント出力を読み取れないことに注意してください。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_PRETTY_PRINT**	boolean	**false**
quarkus.log.syslog.json.date-format 使用する日付フォーマット。"default" は特別な文字列で、デフォルトフォーマットを使用する必要があることを示します。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_DATE_FORMAT**	string	**default**
quarkus.log.syslog.json.record-delimiter 使用する特別なレコード終了区切り文字。デフォルトでは、改行が使用されます。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_RECORD_DELIMITER**	string	
quarkus.log.syslog.json.zone-id 使用するゾーン ID。"default" は特別な文字列で、デフォルトのゾーンを使用する必要があることを示します。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_ZONE_ID**	string	**default**
quarkus.log.syslog.json.exception-output-type 指定する例外出力型。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_EXCEPTION_OUTPUT_TYPE**	**detailed**, **formatted**, **detailed-and-formatted**	**detailed**

quarkus.log.syslog.json.print-details ログでの詳細出力を有効にします。 値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。詳細には、ソースクラス名、ソースファイル名、ソースメソッド名、およびソース行番号が含まれます。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_PRINT_DETAILS**	boolean	**false**
quarkus.log.syslog.json.key-overrides カスタム値でキーをオーバーライドします。この値を省略すると、キーはオーバーライドされません。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_KEY_OVERRIDES**	string	
quarkus.log.syslog.json.excluded-keys JSON 出力から除外するキー。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_EXCLUDED_KEYS**	文字列のリスト	
quarkus.log.syslog.json.additional-field."field-name".value 追加フィールドの値。 環境変数: **QUARKUS_LOG_SYSLOG_JSON_ADDITIONAL_FIELD__FIELD_NAME__VALUE**	string	required
quarkus.log.syslog.json.additional-field."field-name".type 追加フィールドの型指定。サポートされている型: string、int、long (指定されていない場合は string がデフォルト) 環境変数: **QUARKUS_LOG_SYSLOG_JSON_ADDITIONAL_FIELD__FIELD_NAME__TYPE**	**string**, **int**, **long**	**string**

警告

プリティープリントを有効にすると、特定のプロセッサーや JSON パーサーが失敗する場合があります。



注記

値は呼び出し元から取得されるため、詳細の出力はリソースを多く使用する可能性があります。詳細には、ソースクラス名、ソースファイル名、ソースメソッド名、およびソース行番号が含まれます。

1.6. ログハンドラー

ログハンドラーは、ログイベントを受信者に送信するロギングコンポーネントです。Quarkus には、**console**、**file**、**syslog** など、いくつかのログハンドラーが含まれています。

ここで示す例では、ログカテゴリーとして **com.example** を使用しています。

1.6.1. コンソールログハンドラー

コンソールログハンドラーはデフォルトで有効になっており、すべてのログイベントをアプリケーションのコンソール (通常はシステムの **stdout)** に送信します。

- グローバル設定の例:

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

- カテゴリーごとの設定の例:

```
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
```

設定の詳細は、コンソールロギング設定 リファレンス を参照してください。

1.6.2. ファイルログハンドラー

アプリケーションのホスト上にあるファイルにイベントを記録するには、Quarkus ファイルログハンドラーを使用します。ファイルログハンドラーはデフォルトで無効になっており、使用する場合はまず有効にする必要があります。

Quarkus ファイルログハンドラーは、ログファイルのローテーションをサポートします。ログファイルのローテーションにより、指定された数のバックアップログファイルを維持すると同時に、プライマリーログファイルを管理可能かつ最新の状態に保つことで、長期にわたるログファイル管理を効果的に行えます。

ログファイルのローテーションにより、指定された数のバックアップログファイルを維持すると同時に、プライマリーログファイルを管理可能かつ最新の状態に保つことで、長期にわたるログファイル管理を効果的に行えます。

- グローバル設定の例:

```
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
```

- カテゴリーごとの設定の例:

```
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
```

設定の詳細は、ファイルロギング設定 リファレンス を参照してください。

1.6.3. Syslog ログハンドラー

Quarkus の syslog ハンドラーは、UNIX 系システムでログメッセージの送信に使用される Syslog プロトコルに準じます。これは、RFC 5424 で定義されたプロトコルを利用します。

syslog ハンドラーは、デフォルトで無効になっています。有効にすると、すべてのログイベントが syslog サーバー (通常はアプリケーションのローカル syslog サーバー) に送信されます。

- グローバル設定の例:

```
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
```

- カテゴリーごとの設定の例:

```
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
```

設定の詳細は、Syslog ロギング設定 リファレンス を参照してください。

1.7. ログハンドラーにロギングフィルターを追加する

コンソールログハンドラーなどのログハンドラーは、ログレコードをログに記録するかどうかを決定する フィルター にリンクできます。

ログフィルターを登録するには、以下を実行します。

1. **java.util.logging.Filter** を実装する **final** クラスにアノテーション **@io.quarkus.logging.LoggingFilter** を追加し、**name** プロパティーを設定します。

   フィルターの作成例:

```
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);
    }
}
```

この例では、特定のテキストを含むログレコードをコンソールログから除外しています。フィルターする特定のテキストはハードコーディングされず、**my-filter.part** 設定プロパティーから読み込まれます。

application.properties でフィルターを設定する例:

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

2. **application.properties** にある **filter** 設定プロパティーを使用して、対応するハンドラーにフィルターを割り当てます。

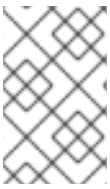
```
quarkus.log.console.filter=my-filter
```

1.8. ロギング設定の例

次の例は、Quarkus でのロギング設定方法を一部示しています。

Quarkus ログ(INFO) 以外の コンソール DEBUG ロギング、色なし、時間短縮、カテゴリー接頭辞の短縮

```
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
```



注記

これらのプロパティーをコマンドラインに追加する場合は、**"** がエスケープされていることを確認してください。たとえば、**-Dquarkus.log.category.\"io.quarkus\".level=DEBUG** です。

ファイル TRACE ロギング設定

```
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
```

注記

ルートロガーを変更しないため、コンソールログには **INFO** 以上のレベルのログのみが含まれます。

カテゴリーに割り当てられた名前付きハンドラー

```
# 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,STRUCTURED_LOGGING_FILE
```

ルートロガーに割り当てられた名前付きハンドラー

```
# 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. ログの一元管理

場所を一元化することで、アプリケーションのさまざまなコンポーネントやインスタンスからログデータを効率的に収集、保存、分析できます。

Graylog、Logstash、Fluentd などの集中化ツールにログを送信する場合は、Quarkus ログの一元管理ガイドを参照してください。

1.10. @QUARKUSTEST のロギング設定

@QuarkusTest の適切なロギングを有効化するには、**java.util.logging.manager** システムプロパティーを **org.jboss.logmanager.LogManager** に設定します。

システムプロパティーを有効にするには、早い段階で設定する必要があるため、ビルドシステム内で設定することが推奨されます。

Maven Surefire プラグイン設定で java.util.logging.manager システムプロパティーを設定する

```
<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. 必ず **org.jboss.logmanager.LogManager** を使用してください。
2. すべてのロギングカテゴリーのデバッグロギングを有効にします。

Gradle の場合、次の設定を **build.gradle** ファイルに追加します。

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

Running **@QuarkusTest** from an IDE も参照してください。

1.11. 他のロギング API を使用する

Quarkus は、すべてのロギング要件に関して JBoss Logging ライブラリーに依存します。

Apache Commons Logging、Log4j、SLF4J などの他のロギングライブラリーに依存するライブラリーを使用すると想定します。その場合は、それらを依存関係から除外し、いずれかの JBoss Logging アダプターを使用します。

ネイティブ実行可能ファイルのコンパイル時に以下のような問題が発生する可能性があるため、これはネイティブ実行可能ファイルをビルドする場合に特に重要です。

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

ロギングの実装はネイティブ実行可能ファイルに含まれませんが、JBoss Logging アダプターを使用してこの問題を解決できます。

次の章で説明するように、これらのアダプターは一般的なオープンソースのロギングコンポーネントで利用できます。

1.11.1. アプリケーションにロギングアダプターを追加する

jboss-logging 以外の各ロギング API では、以下を実行します。

1. ロギングアダプターライブラリーを追加して、これらの API 経由でログに記録されたメッセージが JBoss Log Manager バックエンドにルーティングされるようにします。

注記

この手順は、エクステンションが自動的に処理する Quarkus エクステンションの依存関係であるライブラリーには不要です。

- Apache Commons Logging:
  - Maven の使用:

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

  - Gradle を使用する場合:

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

- Log4j:
  - Maven の使用:

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

  - Gradle を使用する場合:

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

- Log4j 2:
  - Maven の使用:

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

  - Gradle を使用する場合:

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

注記

log4j2-jboss-logmanager ライブラリーには、Log4j をロギング実装として使用するために必要なものがすべて含まれているため、Log4j の依存関係は含めないでください。

- SLF4J:
  - Maven の使用:

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

- Gradle を使用する場合:

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

2. 追加されたライブラリーが生成したログが、他の Quarkus ログと同じフォーマットになっているか確認します。

1.11.2. MDC を使用してコンテキストログ情報を追加する

Quarkus は、リアクティブコアとの互換性を高めるため、ロギングのマッピングされた診断コンテキスト (MDC) をオーバーライドします。

1.11.2.1. MDC データの追加と読み取り

MDC にデータを追加し、ログ出力で展開するには以下を実行します。

1. **MDC** クラスを使用してデータを設定します。
2. ログフォーマットを、**%X{mdc-key}** を使用するようにカスタマイズします。

次のコードを見てみましょう。

JBoss Logging と io.quarkus.logging.Log の例

```
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!";
    }
}
```

次の行を使用してログフォーマットを設定したとします。

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

MDC データを含むメッセージを取得します。

```
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 とサポートされるロギング API

使用する API に応じて、MDC クラスは若干異なります。ただし、API は非常に似ています。

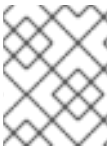
- Log4j 1 - **org.apache.log4j.MDC.put(key, value)**
- Log4j 2 - **org.apache.logging.log4j.ThreadContext.put(key, value)**
- SLF4J - **org.slf4j.MDC.put(key, value)**

1.11.2.3. MDC の伝播

Quarkus では、MDC プロバイダーにはリアクティブコンテキストを処理するための特定の実装があるため、MDC データはリアクティブな非同期処理中に確実に伝播されます。

その結果、さまざまなシナリオで引き続き MDC データにアクセスできます。

- 非同期呼び出しの後、たとえば REST クライアントが Uni を返した場合。
- **org.eclipse.microprofile.context.ManatedExecutor** に送信されたコード内にて。
- **vertx.executeBlocking()** で実行されるコード内にて。



注記

該当する場合、MDC データは **複製されたコンテキスト** に保存されます。これは、単一タスク (リクエスト) を処理するための分離されたコンテキストです。

1.12. ロギング設定リファレンス

ビルド時に固定された設定プロパティー: その他の設定プロパティーはすべてランタイム時にオーバーライド可能

設定プロパティー	型	デフォルト

quarkus.log.level ルートカテゴリーのログレベル。すべてのカテゴリーのデフォルトログレベルとして使用されます。 JBoss Logging は、Apache スタイルのログレベルをサポートします。 ● {@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} 標準の JDK ログレベルもサポートします。 環境変数: **QUARKUS_LOG_LEVEL**	レベル	**INFO**
quarkus.log.handlers ルートカテゴリーにリンクする追加のハンドラーの名前。これらのハンドラーは、consoleHandlers、fileHandlers、または syslogHandlers で定義されます。 環境変数: **QUARKUS_LOG_HANDLERS**	文字列のリスト	
コンソールロギング	**型**	**デフォルト**
quarkus.log.console.enable コンソールログの有効化が必要かどうか 環境変数: **QUARKUS_LOG_CONSOLE_ENABLE**	boolean	**true**
quarkus.log.console.stderr コンソールロギングを、**System#out** ではなく **System#err** にする場合。 環境変数: **QUARKUS_LOG_CONSOLE_STDERR**	boolean	**false**

quarkus.log.console.format ログフォーマット。コンソールフォーマットを制御するエクステンション (XML または JSON フォーマットのエクステンションなど) が存在する場合、この値は無視されることに注意してください。 環境変数: **QUARKUS_LOG_CONSOLE_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.console.level コンソールのログレベル。 環境変数: **QUARKUS_LOG_CONSOLE_LEVEL**	レベル	**ALL**
quarkus.log.console.darken 色をどの程度暗くするかを指定します。コンソールフォーマットを制御するエクステンション (XML または JSON フォーマットのエクステンションなど) が存在する場合、この値は無視されることに注意してください。 環境変数: **QUARKUS_LOG_CONSOLE_DARKEN**	int	**0**
quarkus.log.console.filter コンソールハンドラーにリンクするフィルターの名前。 環境変数: **QUARKUS_LOG_CONSOLE_FILTER**	string	
quarkus.log.console.async 非同期でログを記録するかどうかを示します。 環境変数: **QUARKUS_LOG_CONSOLE_ASYNC**	boolean	**false**
quarkus.log.console.async.queue-length 書き込みをフラッシュする前に使用するキューの長さ。 環境変数: **QUARKUS_LOG_CONSOLE_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.console.async.overflow キューが制限に達した場合に、(メッセージをドロップするのではなく) パブリッシャーをブロックするかどうかを決定します。 環境変数: **QUARKUS_LOG_CONSOLE_ASYNC_OVERFLOW**	**block**, **discard**	**block**

File logging	型	デフォルト
quarkus.log.file.enable ファイルロギングを有効にする必要があるかどうか。 環境変数: **QUARKUS_LOG_FILE_ENABLE**	boolean	**false**
quarkus.log.file.format ログフォーマット 環境変数: **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 ファイルに書き込まれるログのレベル。 環境変数: **QUARKUS_LOG_FILE_LEVEL**	レベル	**ALL**
quarkus.log.file.path ログが書き込まれるファイルの名前。 環境変数: **QUARKUS_LOG_FILE_PATH**	ファイル	**quarkus.log**
quarkus.log.file.filter ファイルハンドラーにリンクするフィルターの名前。 環境変数: **QUARKUS_LOG_FILE_FILTER**	string	
quarkus.log.file.encoding 使用される文字エンコーディング 環境変数: **QUARKUS_LOG_FILE_ENCODING**	Charset	
quarkus.log.file.async 非同期でログを記録するかどうかを示します。 環境変数: **QUARKUS_LOG_FILE_ASYNC**	boolean	**false**

quarkus.log.file.async.queue-length 書き込みをフラッシュする前に使用するキューの長さ。 環境変数: **QUARKUS_LOG_FILE_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.file.async.overflow キューが制限に達した場合に、(メッセージをドロップするのではなく) パブリッシャーをブロックするかどうかを決定します。 環境変数: **QUARKUS_LOG_FILE_ASYNC_OVERFLOW**	**block**, **discard**	**block**
quarkus.log.file.rotation.max-file-size 到達した場合にローテーションが実行されるログファイル最大サイズ。 環境変数: **QUARKUS_LOG_FILE_ROTATION_MAX_FILE_SIZE**	MemorySize	**10 M**
quarkus.log.file.rotation.max-backup-index 保持するバックアップの最大数。 環境変数: **QUARKUS_LOG_FILE_ROTATION_MAX_BACKUP_INDEX**	int	**5**
quarkus.log.file.rotation.file-suffix ファイルハンドラーローテーションファイルの接尾辞。使用すると、ファイルはその接尾辞に基づきローテーションされます。 fileSuffix の例: .yyyy-MM-dd 注記: 接尾辞が .zip または .gz で終わる場合、ローテーションファイルも圧縮されます。 環境変数: **QUARKUS_LOG_FILE_ROTATION_FILE_SUFFIX**	string	
quarkus.log.file.rotation.rotate-on-boot サーバーの初期化時にログファイルをローテーションするかどうかを示します。 これが機能するには、**max-file-size** か **file-suffix** を設定する必要があります。 環境変数: **QUARKUS_LOG_FILE_ROTATION_ROTATE_ON_BOOT**	boolean	**true**
Syslog logging	**型**	**デフォルト**
quarkus.log.syslog.enable syslog ロギングを有効にする必要があるかどうか。 環境変数: **QUARKUS_LOG_SYSLOG_ENABLE**	boolean	**false**

quarkus.log.syslog.endpoint Syslog サーバーの IP アドレスとポート。 環境変数: **QUARKUS_LOG_SYSLOG_ENDPOINT**	host:port	**localhost:514**
quarkus.log.syslog.app-name メッセージを RFC5424 フォーマットでフォーマットするときに使用されるアプリケーション名。 環境変数: **QUARKUS_LOG_SYSLOG_APP_NAME**	string	
quarkus.log.syslog.hostname メッセージ送信元のホスト名。 環境変数: **QUARKUS_LOG_SYSLOG_HOSTNAME**	string	

quarkus.log.syslog.facility RFC-5424 および RFC-3164 で定義されるとおり、メッセージの優先度を計算する際に使用するファシリティーを設定します。 環境変数: **QUARKUS_LOG_SYSLOG_FACILITY**	**kernel**, **user-level**, **mail-system**, **system-daemons**, **security**, **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 このハンドラーが送信メッセージのフォーマットに使用する **SyslogType syslog type** を設定します。 環境変数: **QUARKUS_LOG_SYSLOG_SYSLOG_TYPE**	**rfc5424**, **rfc3164**	**rfc5424**
quarkus.log.syslog.protocol Syslog サーバーへの接続に使用するプロトコルを設定します。 環境変数: **QUARKUS_LOG_SYSLOG_PROTOCOL**	**tcp**, **udp**, **ssl-tcp**	**tcp**
quarkus.log.syslog.use-counting-framing 有効にすると、送信されるメッセージの先頭にメッセージのサイズが付加されます。 環境変数: **QUARKUS_LOG_SYSLOG_USE_COUNTING_FRAMING**	boolean	**false**
quarkus.log.syslog.truncate **true** に設定すると、メッセージが最大長を超えた場合に切り捨てます。 環境変数: **QUARKUS_LOG_SYSLOG_TRUNCATE**	boolean	**true**
quarkus.log.syslog.block-on-reconnect **org.jboss.logmanager.handlers.SyslogHandler.Protocol#TCP TCP** または **org.jboss.logmanager.handlers.SyslogHandler.Protocol#SSL_TCP SSL TCP** プロトコルへの再接続を試みた場合のブロックを有効または無効にします。 環境変数: **QUARKUS_LOG_SYSLOG_BLOCK_ON_RECONNECT**	boolean	**false**
quarkus.log.syslog.format ログメッセージのフォーマット。 環境変数: **QUARKUS_LOG_SYSLOG_FORMAT**	string	**%d{yyyy-MM-dd HH:mm:ss,SSS} %-5p [%c{3.}] (%t) %s%e%n**
quarkus.log.syslog.level Syslog ロガーによって記録されるメッセージレベルを指定するログレベル。 環境変数: **QUARKUS_LOG_SYSLOG_LEVEL**	レベル	**ALL**

quarkus.log.syslog.filter ファイルハンドラーにリンクするフィルターの名前。 環境変数: **QUARKUS_LOG_SYSLOG_FILTER**	string	
quarkus.log.syslog.async 非同期でログを記録するかどうかを示します。 環境変数: **QUARKUS_LOG_SYSLOG_ASYNC**	boolean	**false**
quarkus.log.syslog.async.queue-length 書き込みをフラッシュする前に使用するキューの長さ。 環境変数: **QUARKUS_LOG_SYSLOG_ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.syslog.async.overflow キューが制限に達した場合に、(メッセージをドロップするのではなく) パブリッシャーをブロックするかどうかを決定します。 環境変数: **QUARKUS_LOG_SYSLOG_ASYNC_OVERFLOW**	**block**, **discard**	**block**
Logging categories	**型**	**デフォルト**
quarkus.log.category."categories".level このカテゴリーのログレベル。 **INFO** 未満のログレベルを取得するには、ビルド時の最小レベル設定オプションも調整する必要があることに注意してください。 環境変数: **QUARKUS_LOG_CATEGORY__CATEGORIES__LEVEL**	InheritableLevel	**inherit**
quarkus.log.category."categories".handlers このカテゴリーにリンクするハンドラーの名前。 環境変数: **QUARKUS_LOG_CATEGORY__CATEGORIES__HANDLERS**	文字列のリスト	
quarkus.log.category."categories".use-parent-handlers このロガーが出力を親ロガーに送信するかどうかを指定します。 環境変数: **QUARKUS_LOG_CATEGORY__CATEGORIES__USE_PARENT_HANDLERS**	boolean	**true**
コンソールハンドラー	**型**	**デフォルト**

quarkus.log.handler.console."console-handlers".enable コンソールログの有効化が必要かどうか 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ENABLE**	boolean	**true**
quarkus.log.handler.console."console-handlers".stderr コンソールロギングを、**System#out** ではなく **System#err** にする場合。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__STDERR**	boolean	**false**
quarkus.log.handler.console."console-handlers".format ログフォーマット。コンソールフォーマットを制御するエクステンション (XML または JSON フォーマットのエクステンションなど) が存在する場合、この値は無視されることに注意してください。 環境変数: **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 コンソールのログレベル。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__LEVEL**	レベル	**ALL**
quarkus.log.handler.console."console-handlers".darken 色をどの程度暗くするかを指定します。コンソールフォーマットを制御するエクステンション (XML または JSON フォーマットのエクステンションなど) が存在する場合、この値は無視されることに注意してください。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__DARKEN**	int	**0**
quarkus.log.handler.console."console-handlers".filter コンソールハンドラーにリンクするフィルターの名前。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__FILTER**	string	

quarkus.log.handler.console."console-handlers".async 非同期でログを記録するかどうかを示します。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC**	boolean	**false**
quarkus.log.handler.console."console-handlers".async.queue-length 書き込みをフラッシュする前に使用するキューの長さ。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.console."console-handlers".async.overflow キューが制限に達した場合に、(メッセージをドロップするのではなく) パブリッシャーをブロックするかどうかを決定します。 環境変数: **QUARKUS_LOG_HANDLER_CONSOLE__CONSOLE_HANDLERS__ASYNC_OVERFLOW**	**block**, **discard**	**block**
ファイルハンドラー	**型**	**デフォルト**
quarkus.log.handler.file."file-handlers".enable ファイルロギングを有効にする必要があるかどうか。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ENABLE**	boolean	**false**
quarkus.log.handler.file."file-handlers".format ログフォーマット 環境変数: **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 ファイルに書き込まれるログのレベル。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__LEVEL**	レベル	**ALL**
quarkus.log.handler.file."file-handlers".path ログが書き込まれるファイルの名前。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__PATH**	ファイル	**quarkus.log**
quarkus.log.handler.file."file-handlers".filter ファイルハンドラーにリンクするフィルターの名前。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__FILTER**	string	
quarkus.log.handler.file."file-handlers".encoding 使用される文字エンコーディング 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ENCODING**	Charset	
quarkus.log.handler.file."file-handlers".async 非同期でログを記録するかどうかを示します。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC**	boolean	**false**
quarkus.log.handler.file."file-handlers".async.queue-length 書き込みをフラッシュする前に使用するキューの長さ。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.file."file-handlers".async.overflow キューが制限に達した場合に、(メッセージをドロップするのではなく) パブリッシャーをブロックするかどうかを決定します。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ASYNC_OVERFLOW**	**block**, **discard**	**block**
quarkus.log.handler.file."file-handlers".rotation.max-file-size 到達した場合にローテーションが実行されるログファイル最大サイズ。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_MAX_FILE_SIZE**	MemorySize	**10 M**

quarkus.log.handler.file."file-handlers".rotation.max-backup-index 保持するバックアップの最大数。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_MAX_BACKUP_INDEX**	int	**5**
quarkus.log.handler.file."file-handlers".rotation.file-suffix ファイルハンドラーローテーションファイルの接尾辞。使用すると、ファイルはその接尾辞に基づきローテーションされます。 fileSuffix の例: .yyyy-MM-dd 注記: 接尾辞が .zip または .gz で終わる場合、ローテーションファイルも圧縮されます。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_FILE_SUFFIX**	string	
quarkus.log.handler.file."file-handlers".rotation.rotate-on-boot サーバーの初期化時にログファイルをローテーションするかどうかを示します。 これが機能するには、**max-file-size** か **file-suffix** を設定する必要があります。 環境変数: **QUARKUS_LOG_HANDLER_FILE__FILE_HANDLERS__ROTATION_ROTATE_ON_BOOT**	boolean	**true**
Syslog ハンドラー	**型**	**デフォルト**
quarkus.log.handler.syslog."syslog-handlers".enable syslog ロギングを有効にする必要があるかどうか。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ENABLE**	boolean	**false**
quarkus.log.handler.syslog."syslog-handlers".endpoint Syslog サーバーの IP アドレスとポート。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ENDPOINT**	host:port	**localhost:514**

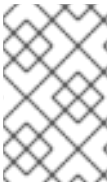
quarkus.log.handler.syslog."syslog-handlers".app-name メッセージを RFC5424 フォーマットでフォーマットするときに使用されるアプリケーション名。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__APP_NAME**	string	
quarkus.log.handler.syslog."syslog-handlers".hostname メッセージ送信元のホスト名。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__HOSTNAME**	string	

quarkus.log.handler.syslog."syslog-handlers".facility RFC-5424 および RFC-3164 で定義されるとおり、メッセージの優先度を計算する際に使用するファシリティーを設定します。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FACILITY**	**kernel**, **user-level**, **mail-system**, **system-daemons**, **security**, **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 このハンドラーが送信メッセージのフォーマットに使用する **SyslogType syslog type** を設定します。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__SYSLOG_TYPE**	**rfc5424**, **rfc3164**	**rfc5424**
quarkus.log.handler.syslog."syslog-handlers".protocol Syslog サーバーへの接続に使用するプロトコルを設定します。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__PROTOCOL**	**tcp**, **udp**, **ssl-tcp**	**tcp**
quarkus.log.handler.syslog."syslog-handlers".use-counting-framing 有効にすると、送信されるメッセージの先頭にメッセージのサイズが付加されます。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__USE_COUNTING_FRAMING**	boolean	**false**
quarkus.log.handler.syslog."syslog-handlers".truncate **true** に設定すると、メッセージが最大長を超えた場合に切り捨てます。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__TRUNCATE**	boolean	**true**
quarkus.log.handler.syslog."syslog-handlers".block-on-reconnect **org.jboss.logmanager.handlers.SyslogHandler.Protocol#TCP TCP** または **org.jboss.logmanager.handlers.SyslogHandler.Protocol#SSL_TCP SSL TCP** プロトコルへの再接続を試みた場合のブロックを有効または無効にします。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__BLOCK_ON_RECONNECT**	boolean	**false**

quarkus.log.handler.syslog."syslog-handlers".format ログメッセージのフォーマット。 環境変数: **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 Syslog ロガーによって記録されるメッセージレベルを指定するログレベル。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__LEVEL**	レベル	**ALL**
quarkus.log.handler.syslog."syslog-handlers".filter ファイルハンドラーにリンクするフィルターの名前。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__FILTER**	string	
quarkus.log.handler.syslog."syslog-handlers".async 非同期でログを記録するかどうかを示します。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC**	boolean	**false**
quarkus.log.handler.syslog."syslog-handlers".async.queue-length 書き込みをフラッシュする前に使用するキューの長さ。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC_QUEUE_LENGTH**	int	**512**
quarkus.log.handler.syslog."syslog-handlers".async.overflow キューが制限に達した場合に、(メッセージをドロップするのではなく) パブリッシャーをブロックするかどうかを決定します。 環境変数: **QUARKUS_LOG_HANDLER_SYSLOG__SYSLOG_HANDLERS__ASYNC_OVERFLOW**	**block**, **discard**	**block**
ログクリーンアプフィルター - 内部使用	**型**	**デフォルト**

quarkus.log.filter."filters".if-starts-with 照合するメッセージの接頭辞。 環境変数: **QUARKUS_LOG_FILTER__FILTERS__IF_STARTS_WITH**	文字列のリスト	**inherit**
quarkus.log.filter."filters".target-level フィルタリングされたメッセージの新しいログレベル。デフォルトは DEBUG です。 環境変数: **QUARKUS_LOG_FILTER__FILTERS__TARGET_LEVEL**	レベル	**DEBUG**



MEMORYSIZE フォーマットについて

サイズ設定オプションは、**[0-9]+[KkMmGgTtPpEeZzYy]?** フォーマット (正規表現として表示) の文字列を認識します。接尾辞が指定されていない場合は、バイトとみなされます。