目次

改訂情報

変更年月日	変更内容
2013-10-01	初版
2014-01-01	第２版　下記を追加・変更しました ■「*セキュリティログ*」の項目に「*ログレベルごとの出力内容*」を追記しました。 ■「*付録*」に「*セットアップ実行ログについて*」を追記しました。 ■「*付録*」に「*インポートエラーの原因をログから追跡する*」を追記しました。
2014-04-01	第３版　下記を追加・変更しました ■「*バーチャルテナントでのログ運用*」の項目を追記しました。 ■「*特定用途ログ*」、「*システムログ*」の「利用可能なMDCキー」の項目にユーザコード、ユーザ種別、テナントID、認証状態を追記しました。
2014-12-01	第４版　下記を追加・変更しました ■「*特定用途ログ*」に「*IM-MessageHub ログ*」を追記しました。
2015-08-01	第５版　下記を追加・変更しました ■「*概要*」の「*SLF4J*」と「*Logback*」、および、「*付録*」の「*commons-logging, log4jの利用について*」のライブラリのバージョン情報を更新しました。 ■「*Appender*」の「*TimeBasedRollingPolicy*」の「maxHistory」の説明を修正しました。 ■「*Appender*」の「*FixedWindowRollingPolicy*」にバージョンごとの最大ファイル数の説明を追記しました。 ■「*SQLログを出力する*」の im_logger_log4jdbc.xml の内容を修正しました。 ■「*特定用途ログ*」に「*テーブルメンテナンス操作ログ*」と「*テーブルメンテナンスインポート・エクスポートログ*」の説明を追記しました。
2015-12-01	第６版　下記を追加・変更しました ■「*システムログ*」に intra-mart Accel Platform 2013 Summer(Damask) 以前のメッセージコード出力について追記しました。 ■リモートアドレス（request.remote.address）、および、リモートホスト（request.remote.host） の出力方式変更に関する説明を追記しました。 　■「*リクエストログ*」の「*利用可能なMDCキー*」 　■「*画面遷移ログ*」の「*利用可能なMDCキー*」 ■「*特定用途ログ*」に「*ユーザコンテキストログ*」、「*EHCache ログ*」の説明を追記しました。 ■「*SQLログを出力する*」に im_logger_mirage.xml の説明を追記しました。 ■「*特定用途ログ*」に「*IM-LogicDesignerログ*」を追記しました。
2016-08-01	第７版　下記を追加・変更しました ■「*概要*」に TERASOLUNA Server Framework for Java (5.x) プログラミングガイドへのリンクを追記しました。
2016-12-01	第８版　下記を変更しました ■DB2に関する記述を削除 ■「*概要*」の「*SLF4J*」と「*Logback*」のライブラリのバージョン情報を更新しました。
2017-04-01	第９版　下記を追加しました ■「*特定用途ログ*」に「*Hazelcastログ*」を追記しました。
2017-12-01	第１０版　下記を追加・変更しました ■「*特定用途ログ*」に「*スマートメニューランキングログ*」を追記しました。 ■「*画面遷移ログ*」の「利用可能なMDCキー」の項目にクライアントタイプを追記しました。
2018-04-01	第１１版　下記を変更しました ■「*セキュリティログ*」の「*ログレベルごとの出力内容*」に2段階認証に関する内容を追記しました。

変更年月日	変更内容
2019-12-01	第１２版　下記を変更しました ■ 「*テーブルメンテナンス操作ログ*」のim_logger_tablemaintenance.xmlの内容について、immediateFlushタグをencoderタグ配下からappenderタグ配下に修正。 ■ 「*テーブルメンテナンスインポート・エクスポートログ*」のim_logger_tablemaintenance.xmlの内容について、immediateFlushタグをencoderタグ配下からappenderタグ配下に修正。 ■ 「*MDCに格納されている値でログを分割する*」の im_logger_security.xml について、immediateFlush タグを削除。 ■ 「*インポート・エクスポートログ*」の im_logger_import_export.xml について、immediateFlush タグを削除。 ■ 「*IM-MessageHub ログ*」の im_logger_message_hub.xml について、immediateFlush タグを削除。 ■ 「*リクエストログ*」の im_logger_request.xml について、immediateFlush タグを削除。 ■ 「*セキュリティログ*」の im_logger_security.xml について、immediateFlush タグを削除。 ■ 「*スマートメニューランキングログ*」の im_logger_smart_menu_ranking.xml について、immediateFlush タグを削除。 ■ 「*テーブルメンテナンス操作ログ*」の im_logger_tablemaintenance_edit.xml について、immediateFlush タグを削除。 ■ 「*テーブルメンテナンスインポート・エクスポートログ*」の im_logger_tablemaintenance_import_export.xml について、immediateFlush タグを削除。 ■ 「*画面遷移ログ*」の im_logger_transition.xml について、immediateFlush タグを削除。 ■ 「*マスタデータ更新ログ*」の im_logger_update_master_data.xml について、immediateFlush タグを削除。 ■ 「*ユーザコンテキストログ*」の im_logger_user_context.xml について、immediateFlush タグを削除。 ■ 「*システムログ*」の im_logger.xml について、immediateFlush タグを削除。 ■ 「*バーチャルテナントでのログ運用*」の im_logger_separate_tenant.xml について、immediateFlush タグを削除。 ■ 「*はじめに*」の 注意事項に、LogbackUtil での Warning ログに関する記述を追加。 ■ 「*Appender*」の encoder プロパティを実装しているクラスに immediateFlush プロパティ を追加 ■ 「*Encoder*」の immediateFlush プロパティに Warning ログに関する記述を追加。

はじめに

本書の目的

本書では intra-mart Accel Platform でのログ機構の詳細について説明します。

説明範囲は以下の通りです。

- intra-mart Accel Platform におけるログの実装
- ログの設定方法

対象読者

本書では次の利用者を対象としています。

- intra-mart Accel Platform の 運用担当者
- アプリケーションでログ出力を行う開発者
- ログによるデバッグを行う開発者

注意事項

1. 本書はアプリケーションが出力するログを対象としています。Webアプリケーションサーバが出力するログに対しては言及していません。
2. 2019 Summer(Waltz) 以降でアップデートを行うと、system.log に LogbackUtil に関する Warning ログが出力されます。warning ログの出力を回避するには、 logger.xml 内の immediateFlush タグを encoder タグの下から appender タグの下へ移動させてください。
   詳細は FAQ を確認ください。

本書の構成

- *概要*

  intra-mart Accel Platform におけるログの概要について説明します。

- *Logger*

  ログ設定ファイルにて利用する、*Logger* の設定方法について説明します。

- *ログクラス紹介*

  ログ設定ファイルにて利用する、*Appender*、*Encoder* などの設定とそのクラスの動作について説明します。

- *システムログ／特定用途ログ*

  intra-mart Accel Platform で標準で提供するログについて説明します。

概要

intra-mart Accel Platform では ロギングライブラリとして 「*SLF4J*」、その実装ライブラリとして「*Logback*」を採用しています。

SLF4J

SLF4J（Simple Logging Facade for Java）は様々なロギングフレームワークのインタフェースを提供します。
これにより、ログを利用するアプリケーションは任意のロギングフレームワークを配置することが可能です。
SLF4Jの詳細な情報については「SLF4JのWebサイト」を参照してください。

intra-mart Accel Platform で利用しているライブラリのバージョン情報は以下の通りです。

iAP の バージョン	バージョン	jarファイル名
2013 Summer(Damask) 以前	1.6.6	slf4j-api-1.6.6.jar
2013 Autumn(Eden) から 2014 Winter(Iceberg) 以前	1.7.5	slf4j-api-1.7.5.jar
2015 Spring(Juno) から 2016 Summer(Nirvana) 以前	1.7.10	slf4j-api-1.7.10.jar
2016 Winter(Olga) 以降	1.7.21	slf4j-api-1.7.21.jar

Logback

LogbackはSLF4Jの実装ライブラリです。
Logbackの詳細な情報については「LogbackのWebサイト」を参照してください。

intra-mart Accel Platform で利用しているライブラリのバージョン情報は以下の通りです。

iAP の バージョン	バージョン	jarファイル名
2013 Summer(Damask) 以前	1.0.7	logback-core-1.0.7.jar logback-classic-1.0.7.jar
2013 Autumn(Eden) から 2014 Winter(Iceberg) 以前	1.0.13	logback-core-1.0.13.jar logback-classic-1.0.13.jar
2015 Spring(Juno) から 2016 Summer(Nirvana) 以前	1.1.2	logback-core-1.1.2.jar logback-classic-1.1.2.jar
2016 Winter(Olga) 以降	1.1.7	logback-core-1.1.7.jar logback-classic-1.1.7.jar

ログ設定

intra-mart Accel Platform では、Webアプリケーション起動時に「*Logback*」に対してログの設定を反映します。
ログの設定は `%CONTEXT_PATH%/WEB-INF/conf/log` ディレクトリ直下の設定ファイルを利用します。具体的には、 `%CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml` を基点として、`%CONTEXT_PATH%/WEB-INF/conf/log` 直下の拡張子が `.xml` であるファイルが統合された内容がログの設定として扱われます。

> **注意**
> 設定ファイルの退避を行う場合は、`%CONTEXT_PATH%/WEB-INF/conf/log` 直下に配置しないように注意してください。

im_logger.xml でのログ設定

`im_logger.xml` では以下のように `<configuration>` タグの中に設定を記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <!-- 省略 -->

</configuration>
```

`im_logger.xml` で可能な設定は主に以下の通りです。

種類	必須設定	複数設定	説明
appender	○	○	ログをどの場所に、どのように、どのような形式で出力するのかを設定します。 詳細については「*Appender*」を参照してください。
root	×	×	ロガー名による設定が何も行われいない場合に利用されるロガーの設定です。 詳細については「*ルートロガー*」を参照してください。

logger	×	○	特定のロガーに対してのどのように出力するかを設定します。 詳細については「*Logger*」を参照してください。

コラム
その他の利用可能な設定については「LogbackのWebサイト」を参照してください。

注意
im_logger.xml の移動や削除は行わないでください。intra-mart Accel Platform でのログ機能が正常に動作しなくなります。

im_logger.xml 以外でのログ設定

im_logger.xml 以外の設定ファイルは、以下のようなタグの中に設定を記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<included>

  <!-- 省略 -->

</included>
```

im_logger.xml 以外の設定ファイルで可能な設定は主に以下の通りです。

種類	必須設定	複数設定	説明
appender	×	○	ログをどの場所に、どのように、どのような形式で出力するのかを設定します。 詳細については「*Appender*」を参照してください。
logger	×	○	特定のロガーに対してのどのように出力するか設定します。 詳細については「*Logger*」を参照してください。

コラム
その他の利用可能な設定については「LogbackのWebサイト」を参照してください。

ログ設定ファイルの編集

intra-mart Accel Platform で提供されているログのログ設定ファイルは、IM-Juggling で編集可能です。同梱されているモジュール名、および、出力場所については「*システムログ*」、および、「*特定用途ログ*」の各ログの項目に記載されています。
本書ではログ設定ファイルのパスは %CONTEXT_PATH%/WEB-INF/conf/log と記載していますが、IM-Juggling でファイルを編集する場合は IM-Juggling のプロジェクト直下の conf/log に対して行ってください。

MDC

MDC（Mapped Diagnostic Context）とは、ログに出力する情報を一時的に保持するマップです。
ログ設定ファイルのLayoutの設定を行うことで特定のキーに保存された情報を出力することが可能です。

intra-mart Accel Platform が提供する各ログで利用可能なMDCについては「*システムログ*」、および、「*特定用途ログ*」の各ログの項を参照してください。

プロパティ

ログ設定ファイルでは ${property_name} と指定することにより様々な設定値を利用可能です。利用可能な設定値は以下のとおりです。

1. ログ設定ファイルの <property> タグで設定したプロパティ
2. Javaのシステムプロパティ（java.lang.System#getProperty(String) で取得できる値）
3. OSの環境変数

コラム
プロパティ名が重複した場合、上記の順番が低い方が優先されます。

ログ設定ファイルでプロパティの設定を行うには以下のように記述します。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <property name="name" value="value" />

</configuration>
```

プロパティが存在しない場合は、プロパティ名に _IS_UNDEFINED という文字列が付与された値が利用されます。
例）

`${name}` を指定し、`name` のプロパティが存在しない場合。

```
name_IS_UNDEFINED
```

`${property_name:-property_default}` という形式で指定することでプロパティが存在しない場合に代替する文字列を設定可能です。
例）
`${name:-unknown}` を指定し、`name` のプロパティが存在しない場合。

```
unknown
```

提供されているプロパティ

intra-mart Accel Platform で提供しているプロパティは以下の通りです。

プロパティ名	説明
im.log	intra-mart Accel Platform でログを出力する基点とするディレクトリのパスが取得できます。 この値はサーバコンテキスト設定のログファイルディレクトリで指定されている値が取得できます。 詳細は設定ファイルリファレンスの「サーバコンテキスト設定」を参照してください。

業務処理からのログ出力

intra-mart Accel Platform の業務処理中にログを出力する方法については以下のドキュメントを参照してください。

- 「 SAStruts+S2JDBC プログラミングガイド 」-「 ログ 」
- 「 スクリプト開発モデル プログラミングガイド 」-「 ログ 」
- 「 TERASOLUNA Server Framework for Java (5.x) プログラミングガイド 」-「 ログ 」

Logger

Loggerは、以下の役割を担います。

- ログの出力
- パラメータの出力
- *ログレベル* の指定
- *ロガー名* の指定

Loggerは <logger> で設定します。
<logger> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
name	○	「*ロガー名*」を指定します。
level	×	「*ログレベル*」を指定します。 未指定の場合は、継承元のロガーのログレベルが設定されます。
additivity	×	継承元の（appender-ref などの）設定を受け継ぐかどうかを指定します。 継承元の設定を受け継ぐ場合は、true、受け継がない場合は、false を指定します。 未指定の場合は、true を設定したものとして扱われます。

<logger> で指定可能な子要素は以下の通りです。

子要素名	タイプ	必須設定	説明
level	String	×	「*ログレベル*」を指定します。 未指定の場合は、継承元のロガーのログレベルが設定されます。
appender-ref	String	×	ロガーで利用する「*Appender*」を指定します。Appenderの設定で指定した識別子を設定します。 この要素は、複数設定可能です。

> **コラム**
> level 属性と level 子要素を両方指定した場合は、子要素の設定が優先されます。

> **注意**
> 利用するAppenderの設定は <logger> より上部に記述してください。
> 指定可能なAppenderの設定は、im_logger.xml に記載されているもの、または、同一ファイルに設定されているAppenderのみです。

ルートロガー

ルートロガーとは、ロガー名の設定に一致するロガー名が存在しない場合に利用されるロガーです。
<root> にてルートロガーの設定を行います。
<root> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
level	×	「*ログレベル*」を指定します。 属性、子要素ともに未指定の場合は、DEBUG を設定したものとして扱われます。

<root> で指定可能な子要素は以下の通りです。

子要素名	タイプ	必須設定	説明
level	String	×	「*ログレベル*」を指定します。 属性、子要素ともに未指定の場合は、DEBUG を設定したものとして扱われます。

> **コラム**
> level 属性と level 子要素を両方指定した場合は、子要素の設定が優先されます。

> **注意**
> 利用するAppenderの設定は <root> より上部に記述してください。
> 指定可能なAppenderの設定は、im_logger.xml に記載されているAppenderのみです。

ログレベル

ログ出力時にログの重要度を指定します。指定可能なログレベル、および、用途は以下の通りです。

ログレベル	用途
ERROR	予期しない動作などにより、処理を継続できない場合。アプリケーションでエラーが発生した場合。
WARN	問題が発生したが、処理の継続が可能である場合。運用者によるリカバリが可能である場合。
INFO	運用者に対して障害情報ではない何らかの情報を通知したい場合。
DEBUG	バグ解析などを目的としたデバッグ情報を通知したい場合。
TRACE	メソッドの引数、および、戻り値等、メソッド内の情報を通知したい場合。

ログの設定で利用可能なログレベルの設定値、および、出力範囲は以下の通りです。

設定値	出力範囲
OFF	なにも出力しません。
ERROR	ERROR
WARN	ERROR, WARN
INFO	ERROR, WARN, INFO
DEBUG	ERROR, WARN, INFO, DEBUG
TRACE	ERROR, WARN, INFO, DEBUG, TRACE
ALL	ERROR, WARN, INFO, DEBUG, TRACE
NULL	継承元のロガーと同じ出力範囲が設定されます。ルートロガーでは指定できません。
INHERITED	継承元のロガーと同じ出力範囲が設定されます。ルートロガーでは指定できません。

コラム
設定値として指定するログレベルは大文字、小文字を問いません。

ロガー名

ロガー名とは、ロガーに割り当てる名称です。単純にロガーとも呼びます。ログの出力設定は、ロガーごとに独立して設定することが可能です。
ロガー名は、階層構造を持たせることが可能となっており、ドット（.）で区切られた文字列を指定することで、親階層のロガー名の設定を継承できます。
ルートロガーの設定と各ロガー名の設定により、ログの出力設定が確定します。

ロガー名の設定の継承例

以下では、凡例を用いて階層構造によるロガー名の設定の継承を説明します。

Case 1

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <!-- appender設定は省略しています。 -->

  <root>
    <level value="debug" />
    <appender-ref ref="STDOUT" />
  </root>

</configuration>
```

上記のように %CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml で設定が行われている場合のログ出力は以下の通りです。
ロガー名に対して特に設定を行っていないため、ログがルートロガーの設定で出力されます。

ロガー名	有効となるログレベル	出力先
foo.App	DEBUG	STDOUT
foo.bar.App	DEBUG	STDOUT
foo.bar.baz.App	DEBUG	STDOUT

Case 2

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <!-- appender設定は省略しています。 -->

  <logger name="foo.bar">
    <level value="info" />
    <appender-ref ref="FILE" />
  </logger>

  <root>
    <level value="debug" />
    <appender-ref ref="STDOUT" />
  </root>

</configuration>
```

上記のように %CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml で設定が行われている場合のログ出力は以下の通りです。
foo.bar に対して設定を行っているため、継承関係にある foo.bar.APP、および、foo.bar.baz.App は foo.bar に対しての設定が反映されています。

ロガー名	有効となるログレベル	出力先
foo.App	DEBUG	STDOUT
foo.bar.App	INFO	STDOUT, FILE
foo.bar.baz.App	INFO	STDOUT, FILE

Case 3

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <!-- appender設定は省略しています。 -->

  <logger name="foo.bar">
    <level value="info" />
    <appender-ref ref="FILE" />
  </logger>

  <logger name="foo.bar.baz">
    <level value="debug" />
  </logger>

  <root>
    <level value="warn" />
    <appender-ref ref="STDOUT" />
  </root>

</configuration>
```

上記のように %CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml で設定が行われている場合のログ出力は以下の通りです。
foo.bar.baz.App には、一番近い親である foo.bar.baz の設定が反映されています。

ロガー名	有効となるログレベル	出力先
foo.App	WARN	STDOUT
foo.bar.App	INFO	STDOUT, FILE
foo.bar.baz.App	DEBUG	STDOUT, FILE

Case 4

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <!-- appender設定は省略しています。 -->

  <logger name="foo.bar" additivity="false">
    <level value="info" />
    <appender-ref ref="FILE" />
  </logger>

  <logger name="foo.bar.baz">
    <level value="warn" />
  </logger>

  <root>
    <level value="debug" />
    <appender-ref ref="STDOUT" />
  </root>

</configuration>
```

上記のように %CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml で設定が行われている場合のログ出力は以下の通りです。
foo.bar の additivity 属性に false が指定されているため、foo.bar 配下はルートロガーの設定が反映されません。

ロガー名	有効となるログレベル	出力先
foo.App	DEBUG	STDOUT
foo.bar.App	INFO	FILE
foo.bar.baz.App	WARN	FILE

Appender

Appenderは、以下の役割を担います。

- ログの出力先の決定

Appenderは設定ファイルに `<appender>` で設定します。
`im_logger.xml` に記載する場合は以下のとおりです。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder>
      <outputPatternAsHeader>true</outputPatternAsHeader>
      <pattern>[%level] %logger{10} - %msg%n</pattern>
    </encoder>
  </appender>

</configuration>
```

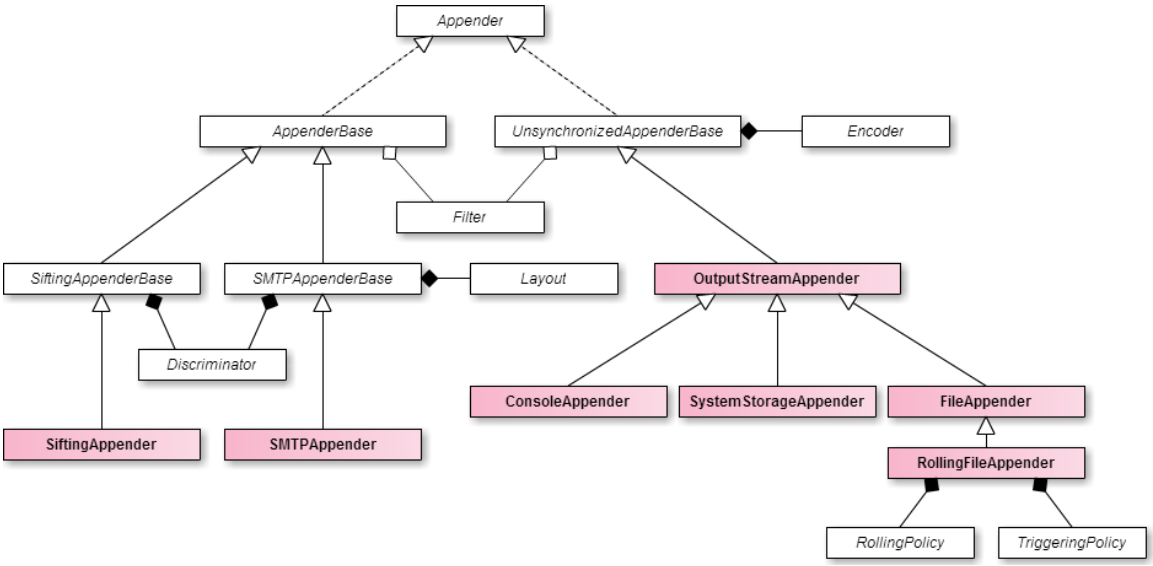
この例では、識別子 STDOUT に対して「*ConsoleAppender*」を割り当てています。

<appender> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
name	○	このAppenderに対しての識別子を指定します。 「*Logger*」ではここで指定した識別子を指定して利用するAppenderを設定します。
class	○	Appenderの完全修飾クラス名を指定します。

Appenderに対しての設定は <appender> の子要素で指定します。設定可能な値（プロパティ）は、Appenderの実装により異なります。

Appenderの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



以下では各種Appenderの実装を紹介します。

> **コラム**
> 本書に記述されていないAppenderについては「LogbackのWebサイト」を参照してください。

ConsoleAppender

ログをコンソールに出力するAppenderです。

クラス名

ch.qos.logback.core.ConsoleAppender

プロパティ

プロパティ名	タイプ	必須設定	説明
filter	*Filter*	×	「*Filter*」を指定します。 <filter> を複数記述することで複数指定可能です。
encoder	*Encoder*	○	「*Encoder*」を指定します。 <encoder> の class 属性が未指定の場合は、「*PatternLayoutEncoder*」を利用します。
target	String	×	出力先を指定します。指定可能な出力先は以下の通りです。 ▪ System.out：標準出力 ▪ System.err：標準エラー

immediateFlush	boolean	×	バッファリングしている内容を即時書き込むかを指定します。 未指定の場合は、true を設定したものとして扱われます。 2019 Spring(Violette) 以前のバージョンでは、immediateFlush タグを appender タグの下から encoder タグの下へ移動させてください。 詳細は FAQ を確認ください。

FileAppender

ログをファイルに出力するためのAppenderです。

> **コラム**
> ログのローテートを行う場合は「*RollingFileAppender*」を利用してください。

クラス名

ch.qos.logback.core.FileAppender

プロパティ

プロパティ名	タイプ	必須設定	説明
filter	*Filter*	×	「*Filter*」を指定します。 <filter> を複数記述することで複数指定可能です。
append	boolean	×	ログ出力開始時に既にファイルが存在した場合に追記を行うかを指定します。 true を指定した場合は、存在するファイルの末尾からログを追記します。 false を指定した場合は、ファイルを削除し、新規ファイルにログを出力します。 未指定の場合は、true を設定したものとして扱われます。
encoder	*Encoder*	○	「*Encoder*」を指定します。 <encoder> の class 属性が未指定の場合は、「*PatternLayoutEncoder*」を利用します。
file	String	○	ファイルの出力先を指定します。指定したディレクトリ／ファイルが存在しない場合は、自動的に生成されます。 Windows環境で \ を利用してファイルパスを指定する場合は、エスケープを行ってください。 例） C:\var\log\system.log にファイルを出力する場合は、 C:\\var\\log\\system.log と指定します。
prudent	boolean	×	prudent モードでログ出力を行うかを指定します。prudent モードを利用した場合、排他ロックを行いながらログの出力を行うため、異なるJVMからの同一ファイルへ安全に書き込むことが可能です。ただし、排他ロックを行うため、パフォーマンスの低下につながることに注意してください。 未指定の場合は、false を設定したものとして扱われます。
immediateFlush	boolean	×	バッファリングしている内容を即時書き込むかを指定します。 未指定の場合は、true を設定したものとして扱われます。 2019 Spring(Violette) 以前のバージョンでは、immediateFlush タグを appender タグの下から encoder タグの下へ移動させてください。 詳細は FAQ を確認ください。

> **注意**
> パラメータ prudent に true を指定した場合は パラメータ append は強制的に true を設定したものとして扱われます。

RollingFileAppender

ログファイルのローテートを行うAppenderです。

クラス名

ch.qos.logback.core.rolling.RollingFileAppender

プロパティ

プロパティ名	タイプ	必須設定	説明
filter	*Filter*	×	「*Filter*」を指定します。 <filter> を複数記述することで複数指定可能です。

append	boolean	×	ログ出力開始時に既にファイルが存在した場合に追記を行うかを指定します。 true を指定した場合は、存在するファイルの末尾からログを追記します。 false を指定した場合は、ファイルを削除し、新規ファイルにログを出力します。 未指定の場合は、true を設定したものとして扱われます。
encoder	*Encoder*	○	「*Encoder*」を指定します。 <encoder> の class 属性が未指定の場合は、「*PatternLayoutEncoder*」を利用します。
file	String	○	ファイルの出力先を指定します。指定したディレクトリ／ファイルが存在しない場合は、自動的に生成されます。 Windows環境で \ を利用してファイルパスを指定する場合は、エスケープを行ってください。 例） C:\var\log\system.log にファイルを出力する場合は、 C:\\var\\log\\system.log と指定します。
rollingPolicy	*RollingPolicy*	○	「*RollingPolicy*」を指定します。 必須の設定ですが、triggeringPolicy に指定する値によっては指定すべきでは無い場合があります。
triggeringPolicy	*TriggeringPolicy*	○	「*TriggeringPolicy*」を指定します。 必須の設定ですが、rollingPolicy に指定する値によっては指定すべきでは無い場合があります。
prudent	boolean	×	prudent モードでログ出力を行うかを指定します。prudent モードを利用した場合、排他ロックを行いながらログの出力を行うため、異なるJVMからの同一ファイルへ安全に書き込むことが可能です。ただし、排他ロックを行うため、パフォーマンスの低下につながることに注意してください。 未指定の場合は、false を設定したものとして扱われます。
immediateFlush	boolean	×	バッファリングしている内容を即時書き込むかを指定します。 未指定の場合は、true を設定したものとして扱われます。 2019 Spring(Violette) 以前のバージョンでは、immediateFlush タグを appender タグの下から encoder タグの下へ移動させてください。 詳細は FAQ を確認ください。

> **注意**
> パラメータ prudent に true を指定した場合は パラメータ append は強制的に true を設定したものとして扱われます。

SMTPAppender

ログをメールで送信するAppenderです。
詳細はLogbackのWebサイト「SMTPAppender」を参照してください。

クラス名

```
ch.qos.logback.classic.net.SMTPAppender
```

プロパティ

プロパティ名	タイプ	必須設定	説明
filter	*Filter*	×	「*Filter*」を指定します。 <filter> を複数記述することで複数指定可能です。
layout	*Layout*	○	「*Layout*」を指定します。 <layout> の class 属性が未指定の場合は、「*PatternLayout*」を利用します。
smtpHost	String	×	SMTPサーバのホスト名を指定します。 <sessionViaJNDI> が false （未指定も含む）である場合、必ず指定してください。
smtpPort	int	×	SMTPサーバのポート番号を指定します。 未指定の場合は、25 を設定したものとして扱われます。
to	String	×	メールの送信先のアドレスを指定します。 この設定は <to> を複数記述することで複数指定可能です。 <sessionViaJNDI> が false （未指定も含む）である場合、必ず指定してください。
from	String	×	メールの送信元のアドレスを指定します。 未指定の場合は、javax.mail.internet.InternetAddress#getLocalAddress(Session) で取得した値が指定されます。
subject	String	×	メールの件名を指定します。 この項目では「*パターン文字列*」が利用可能です。 未指定の場合は、%logger{20} - %m を設定したものとして扱われます。
discriminator	*Discriminator*	×	「*Discriminator*」を指定します。 <discriminator> の class 属性が未指定の場合は「*DefaultDiscriminator*」を利用します。

evaluator	*EventEvaluator*	×	「*EventEvaluator*」を指定します。 EventEvaluatorを指定することで、メールを送信するタイミングを変更することが可能です。 未指定の場合は、ログレベルが ERROR の時のみメールを送信する「*OnErrorEvaluator*」を設定したものとして扱われます。
cyclicBufferTracker	CyclicBufferTracker	×	CyclicBufferTrackerを指定します。 CyclicBufferTrackerから作成したCyclicBufferにより、循環バッファの動作が決定します。 「*Discriminator*」により分別が行われている場合、CyclicBufferは分別された値ごとに作成されます。 詳細はLogbackのWebサイト「SMTPAppender」を参照してください。 未指定の場合は、循環バッファ数は 256 が指定されます。
username	String	×	平文でのSMTP認証を行う場合に、ユーザ名を指定します。
password	String	×	平文でのSMTP認証を行う場合に、パスワードを指定します。
STARTTLS	boolean	×	STARTLS を利用してSMTPサーバに接続を行うかを指定します。 true の場合、STARTLS を利用します。 false の場合、STARTLS を利用しません。 SSLとは異なり、初回の接続は暗号化されません。 未指定の場合は、false が指定されます。
SSL	boolean	×	SSL を利用してSMTPサーバに接続を行うかを指定します。 true の場合、SSLを利用します。 false の場合、SSLを利用しません。 未指定の場合は、false が指定されます。
charsetEncoding	String	×	メールのメッセージのエンコーディングを指定します。 未指定の場合は、UTF-8 が指定されます。
localhost	String	×	SMTPの HELO コマンドまたは EHLO コマンドで使用するローカルホストのドメイン名を指定します。 未指定の場合は、java.net.InetAddress#getLocalhost().getHostName() で取得した値が指定されます。
asynchronousSending	boolean	×	メールを非同期で送信するかどうかを指定します。 true の場合、メールを非同期で送信します。 false の場合、メールを非同期で送信しません。 非同期で送信する場合、アプリケーションの終了直前に出力されたログが送信されない場合があるので注意してください。 未指定の場合は、true が指定されます。
includeCallerData	boolean	×	呼び出し元の情報を含めるかどうかを指定します。 true の場合、呼び出し元の情報を含めます。 false の場合、呼び出し元の情報を含めません。 未指定の場合は、true が指定されます。
sessionViaJNDI	boolean	×	JNDIを利用したメール送信を行うかどうかを指定します。 true を指定した場合、ネーミング・サービスに紐づいたメール設定で送信を行います。 false を指定した場合、SMTPAppenderへの設定情報を元にメール送信を行います。 未指定の場合は、false が指定されます。
jndiLocation	String	×	JNDIのロケーションを指定します。 JNDIを利用してメール送信を行う場合は、<sessionViaJNDI> に true を指定する必要があります。 JNDIを利用したメール送信を行わない場合は、指定する必要がありません。

SiftingAppender

出力イベント（内容）により出力先を「ふるい」にかけ、分別を行うAppenderです。「ふるい」により、分別された値ごとにAppenderを作成し内包します。
内包するAppenderは <timeout> で指定した時間の間利用されない場合、次に自身のAppender（SiftingAppender）により書き込みを行った際に削除されます。
内包するAppenderの設定は <sift> で行います。<sift> でのAppenderの指定方法については、「*MDCに格納されている値でログを分割する*」を参照してください。
<sift> 内のAppenderでは <discriminator> で「ふるい」にかけられた値はプロパティとして利用可能です。

- 例）「*MDCBasedDiscriminator*」を利用し、<key>application.key</key> と指定した場合、${application.key} と記述することで MDC.get("application.key") で取得された値が利用できます。

プロパティの利用方法、詳細については「*プロパティ*」を参照してください。

> **コラム**
>
> 同名のプロパティが既に存在している（ログ設定ファイルなどで同名のプロパティを定義している）場合でも、ここで利用可能になるプロパティが優先されます。

クラス名

ch.qos.logback.classic.sift.SiftingAppender

プロパティ

プロパティ名	タイプ	必須設定	説明
filter	*Filter*	×	「*Filter*」を指定します。 <filter> を複数記述することで複数指定可能です。
timeout	Duration	×	内包するAppenderのタイムアウト時間を設定します。 指定は数値 ＋ 単位で行います。数値は、小数を利用した指定が可能です。単位は、ミリ秒（millisecond/milliseconds）、秒（second/seconds）、分（minute/minutes）、時（hour/hours）、と、日（day/days）で指定可能です。 例） 30 minutes：３０分 1 hour：１時間 0.5 day：１２時間 未指定の場合は、タイムアウト時間は 30分 が設定されます。
maxAppenderCount	int	×	SiftingAppenderが内包可能なAppenderの数を指定します。 未指定の場合は、Javaの Integer.MAX_VALUE が指定されます。
discriminator	*Discriminator*	○	「*Discriminator*」を指定します。 <discriminator> の class 属性が未指定の場合は、「*MDCBasedDiscriminator*」を利用します。

SystemStorageAppender

ログをシステムストレージに出力するためのAppenderです。

クラス名

jp.co.intra_mart.common.platform.log.appender.SystemStorageAppender

プロパティ

プロパティ名	タイプ	必須設定	説明
filter	*Filter*	×	「*Filter*」を指定します。 <filter> を複数記述することで複数指定可能です。
append	boolean	×	ログ出力開始時に既にファイルが存在した場合に追記を行うかを指定します。 true を指定した場合は、存在するファイルの末尾からログを追記します。 false を指定した場合は、ファイルを削除し、新規ファイルにログを出力します。 未指定の場合は、true が指定されます。
encoder	*Encoder*	○	「*Encoder*」を指定します。 <encoder> の class 属性が未指定の場合は、「*PatternLayoutEncoder*」を利用します。
file	String	○	ファイルの出力先を指定します。指定したディレクトリ／ファイルが存在しない場合は、自動的に生成されます。 システムストレージのルートからの相対パスで指定します。
prudent	boolean	×	prudent モードでログ出力を行うかを指定します。prudent モードを利用した場合、排他ロックを行いながらログの出力を行うため、異なるJVMからの同一ファイルへ安全に書き込むことが可能です。ただし、排他ロックを行うため、パフォーマンスの低下につながることに注意してください。 未指定の場合は、false を設定したものとして扱われます。
immediateFlush	boolean	×	バッファリングしている内容を即時書き込むかを指定します。 未指定の場合は、true を設定したものとして扱われます。 2019 Spring(Violette) 以前のバージョンでは、immediateFlush タグを appender タグの下から encoder タグの下へ移動させてください。 詳細は FAQ を確認ください。

> **注意**
> パラメータ prudent に true を指定した場合は パラメータ append は強制的に true を設定したものとして扱われます。

RollingPolicy

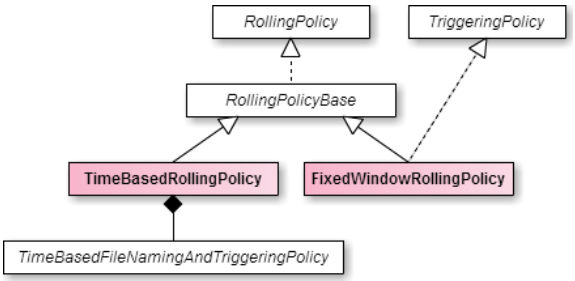
この設定は「*RollingFileAppender*」などの一部のAppenderで利用可能です。<rollingPolicy> で設定します。
RollingPolicyは以下の役割を担います。

- ログのローテート時における、ファイルのローテート方法の決定

<rollingPolicy> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	○	RollingPolicyの実装クラスの完全修飾クラス名を指定します。

RollingPolicyの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



TimeBasedRollingPolicy

日時毎でのログファイルを出力します。

> **注意**
> TimeBasedRollingPolicyは「*TriggeringPolicy*」の機能も兼ね備えています。TimeBasedRollingPolicyをRollingPolicyとして指定する場合は、TriggerPolicyは指定しないでください。

クラス名

`ch.qos.logback.core.rolling.TimeBasedRollingPolicy`

プロパティ

プロパティ名	タイプ	必須設定	説明
fileNamePattern	String	○	バックアップファイルのパターンを指定します。 値に `%d` を指定することで日時を含むパターンを指定することが可能です。また、`%d{` と `}` の間に日時書式形式を指定することが可能です。指定可能な形式は `java.text.SimpleDateFormat` と同様です。 `%d` の後ろに `{}` を指定しなかった場合、`yyyy-MM-dd` 形式として扱われます。 `fileNamePattern` の接尾辞が `.zip`、または、`.gz` の場合、バックアップファイルがそれぞれの形式に圧縮されます。
maxHistory	int	×	バックアップファイルとして残す期間を指定します。 ログ出力時に、指定した期間を超えたログ履歴ファイルは削除されます。 指定する期間の単位は、 `fileNamePattern` の指定に依存します。 例えば、以下の設定がされていた場合、`fileNamePattern` の最小単位が「日」となるため、「10日」が経過したバックアップファイルは削除されます。 ▪ fileNamePattern : %d{yyyy-MM-dd}.log ▪ maxHistory : 10 `0` が指定された場合、無制限にバックアップファイルを残します。 未指定の場合は、`0` を設定したものとして扱われます。
timeBasedFileNamingAndTriggeringPolicy	*TimeBasedFileNamingAndTriggeringPolicy*	×	「*TimeBasedFileNamingAndTriggeringPolicy*」を指定します。 未指定の場合は、 「*DefaultTimeBasedFileNamingAndTriggeringPolicy*」が利用されます。

cleanHistoryOnStart	boolean	×	Appender起動時にバックアップファイルを削除するかどうかを指定します。 true を指定した場合は、存在するバックアップファイルの削除を行います。 false を指定した場合は、バックアップファイルの削除を行いません。 maxHistory の値が 0 の場合は、バックアップファイルの削除を行いません。 未指定の場合は、false を設定したものとして扱われます。

ログの出力例

「*RollingFileAppender*」にて rollingPolicy プロパティに「*TimeBasedRollingPolicy*」を指定した場合のログの出力例を紹介します。
im_logger.xml に以下のような設定が行われていたとします。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <configuration>
3
4    <!-- 省略 -->
5
6    <appender name="SAMPLE" class="ch.qos.logback.core.rolling.RollingFileAppender">
7      <file>${im.log}/platform/sample.log</file>
8      <append>true</append>
9      <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
10       <fileNamePattern>${im.log}/platform/sample-%d{yyyy-MM-dd-HH-mm}.log</fileNamePattern>
11       <maxHistory>2</maxHistory>
12     </rollingPolicy>
13     <encoder>
14       <pattern>%logger{10} - %msg</pattern>
15     </encoder>
16   </appender>
17
18 </configuration>
```

最新のログは、%CONTEXT_PATH%/WEB-INF/log/platform/sample.log に出力され、古いログは sample.log が生成されるタイミング（このサンプルの場合、日時形式に指定されているの最小の時間単位は mm であるため毎分）で sample-yyyy-MM-dd-HH-mm.log というファイル名のバックアップファイルに移されます。
また、maxHistory が 2 であるため、「2分」が経過したログファイルは削除されます。
以下は、実行時間ごとのログファイルとその推移です。（毎分ログ出力が行われている前提とします。）

日時	ログファイル	説明
2013-10-01 00:00	sample.log	sample.log が生成されます。
2013-10-01 00:01	sample.log sample-2013-10-01-00-00.log	sample.log が sample-2013-10-01-00-00.log にリネームされます。 新たに sample.log が生成されます。
2013-10-01 00:02	sample.log sample-2013-10-01-00-00.log sample-2013-10-01-00-01.log	sample.log が sample-2013-10-01-00-01.log にリネームされます。 新たに sample.log が生成されます。
2013-10-01 00:03	sample.log sample-2013-10-01-00-00.log sample-2013-10-01-00-01.log sample-2013-10-01-00-02.log	sample.log が sample-2013-10-01-00-02.log にリネームされます。 新たに sample.log が生成されます。 maxHistory で指定したファイル数を超えるため、sample-2013-10-01-00-00.log が削除されます。

また、ログ出力が実行されない場合は、指定された時間が経過してもログファイルのローテート処理は行われません。
以下では、ログが 00:03 ～ 00:10 の間は出力されず、00:10 以降に出力された場合の推移です。

日時	ログファイル	説明
2013-10-01 00:09	sample.log sample-2013-10-01-00-01.log sample-2013-10-01-00-02.log	ログの出力が行われていないため、ログファイルの状態は変更されません。
2013-10-01 00:10	sample.log sample-2013-10-01-00-01.log sample-2013-10-01-00-02.log sample-2013-10-01-00-09.log	00:10 過ぎにログ出力が実行されたため、 sample.log が sample-2013-10-01-00-09.log にリネームされます。 新たに sample.log が生成されます。 maxHistory で指定したファイル数を超えるため、sample-2013-10-01-00-01.log、および、sample-2013-10-01-00-02.log が削除されます。

FixedWindowRollingPolicy

ローテート時に固定的な数値が付与されたファイル名のバックアップファイル生成します。

クラス名

ch.qos.logback.core.rolling.FixedWindowRollingPolicy

プロパティ

プロパティ名	タイプ	必須設定	説明
fileNamePattern	String	○	バックアップファイルのパターンを指定します。 値に %i を指定することでローテートにより生成された数値を含むパターンを指定することが可能です。 fileNamePattern の接尾辞が .zip、または、.gz の場合、バックアップファイルがそれぞれの形式に圧縮されます。
minIndex	int	×	ローテート時に付与される数値の開始番号を指定します。 未指定の場合は、1 が指定されます。
maxIndex	int	×	ローテート時に付与される数値の終了番号を指定します。 未指定の場合は、7 が指定されます。

注意

生成可能なバックアップファイルの最大数は以下の通りです。
minIndex プロパティと maxIndex プロパティには、最大ファイル数以内となる値を設定してください。

iAP の バージョン	最大ファイル数
2013 Summer(Damask) 以前	13
2013 Autumn(Eden) 以降	21

ログの出力例

「*RollingFileAppender*」にて rollingPolicy プロパティに「*FixedWindowRollingPolicy*」を指定した場合のログの出力例を紹介します。
triggeringPolicy プロパティには「*SizeBasedTriggeringPolicy*」を利用します。
im_logger.xml に以下のような設定が行われていたとします。

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <configuration>
3
4    <!-- 省略 -->
5
6    <appender name="SAMPLE" class="ch.qos.logback.core.rolling.RollingFileAppender">
7      <file>${im.log}/platform/sample.log</file>
8      <append>true</append>
9      <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
10       <fileNamePattern>${im.log}/platform/sample%i.log</fileNamePattern>
11       <minIndex>1</minIndex>
12       <maxIndex>2</maxIndex>
13     </rollingPolicy>
14     <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
15       <maxFileSize>10KB</maxFileSize>
16     </triggeringPolicy>
17     <encoder>
18       <pattern>%logger{10} - %msg</pattern>
19     </encoder>
20   </appender>
21
22 </configuration>
```

最新のログは、%CONTEXT_PATH%/WEB-INF/log/platform/sample.log に出力され、古いログは sample.log が生成されるタイミング（このサンプルの場合、「*SizeBasedTriggeringPolicy*」を利用しているため、sample.log のファイルサイズが10KByteを超える）で sample1.log というファイル名のバックアップファイルに移されます。
以下は、ログファイルとその推移です。

trigger	ログファイル	説明
-	sample.log	sample.log が生成されます。
sample.logローテート時	sample.log sample1.log	sample.log が sample1.log にリネームされます。 新たに sample.log が生成されます。
sample.logローテート時	sample.log sample1.log sample2.log	sample1.log が sample2.log にリネームされます。 sample.log が sample1.log にリネームされます。 新たに sample.log が生成されます。

sample.logローテート時	sample.log sample1.log sample2.log	maxHistory で指定したファイル数を超えるため、sample2.log が削除されます。 sample1.log が sample2.log にリネームされます。 sample.log が sample1.log にリネームされます。 新たに sample.log が生成されます。

TriggeringPolicy

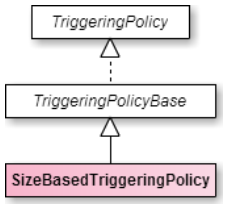
この設定は「*RollingFileAppender*」などの一部のAppenderで利用可能です。<triggeringPolicy> で設定します。
TriggeringPolicyは以下の役割を担います。

- ログのローテートの発生条件の決定

<triggeringPolicy> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	○	TriggeringPolicyの実装クラスの完全修飾クラス名を指定します。

TriggeringPolicyの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



SizeBasedTriggeringPolicy

ローテート元であるログファイルが特定のサイズ以上である場合にローテートを行います。
Appenderによるログ出力時に、以下の条件に一致した場合ローテートを行います。

- ローテート元であるログファイルが特定のサイズ以上である

クラス名

ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy

プロパティ

プロパティ名	タイプ	必須設定	説明
maxFileSize	String	×	最大ファイルサイズを指定します。サイズの指定は、KB、MB、または、GB のいずれかで行う必要があります。 未指定の場合は、10MB を設定したものとして扱われます。

TimeBasedFileNamingAndTriggeringPolicy

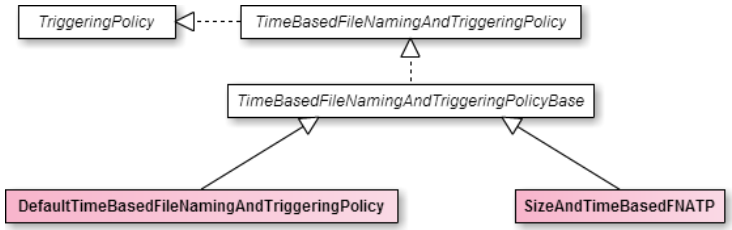
この設定は「*TimeBasedRollingPolicy*」で利用します。<timeBasedFileNamingAndTriggeringPolicy> で設定可能です。
TimeBasedFileNamingAndTriggeringPolicyは以下の役割を担います。

- TimeBasedRollingPolicy利用時のログのローテートの発生条件の決定

<timeBasedFileNamingAndTriggeringPolicy> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	○	TimeBasedFileNamingAndTriggeringPolicyの実装クラスの完全修飾クラス名を指定します。

TimeBasedFileNamingAndTriggeringPolicyの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



DefaultTimeBasedFileNamingAndTriggeringPolicy

TimeBasedFileNamingAndTriggeringPolicyの標準の実装です。TimeBasedRollingPolicyでTimeBasedFileNamingAndTriggeringPolicyの設定を指定しなかった場合に利用されます。
Appenderによるログ出力時に、以下の条件に一致した場合ローテートを行います。

- TimeBasedRollingPolicyで指定したバックアップファイルのパターン（fileNamePattern 設定）に含まれる日時形式の最小単位の時間が、以前ログを出力した時間を超えている

クラス名

ch.qos.logback.core.rolling.DefaultTimeBasedFileNamingAndTriggeringPolicy

プロパティ

設定可能なプロパティはありません。

SizeAndTimeBasedFNATP

日時毎とファイルサイズ毎にローテートを行います。
Appenderによるログ出力時に、以下のいずれかの条件に一致した場合ローテートを行います。

- TimeBasedRollingPolicyで指定したバックアップファイルのパターン（fileNamePattern 設定）に含まれる日時形式の最小単位の時間が、以前ログを出力した時間を超えている
- ローテート元であるログファイルが特定のサイズ以上である

このTimeBasedFileNamingAndTriggeringPolicyを利用する場合は、TimeBasedRollingPolicyの fileNamePattern 設定値 に %i を指定することでローテートにより生成された数値を含むパターンを指定します。
%i の要素数は 0 から始まります。

クラス名

ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP

プロパティ

プロパティ名	タイプ	必須設定	説明
maxFileSize	String	○	最大ファイルサイズを指定します。サイズの指定は、KB、MB、または、GB のいずれかで行う必要があります。

Discriminator

この設定は「*SMTPAppender*」や「*SiftingAppender*」などの一部のAppenderで利用可能です。<discriminator> で設定します。
Discriminatorは、以下の役割を担います。

- ログ出力の分別を行う単位の決定

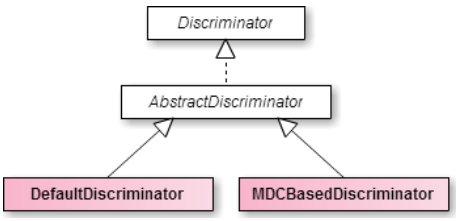
分別された情報の扱い方は、利用するAppenderごとに異なります。

<discriminator> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	×	Discriminatorの実装クラスの完全修飾クラス名を指定します。

> **コラム**
> class 属性省略時に利用されるDiscriminatorの実装クラスについては、各Appenderの設定を参照してください。

Discriminatorの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



DefaultDiscriminator

分別を行わないDiscriminatorの実装です。

クラス名

ch.qos.logback.core.sift.DefaultDiscriminator

プロパティ

設定可能なプロパティはありません。

MDCBasedDiscriminator

<key> で指定したMDCのキーが指す値で分別を行うDiscriminatorの実装です。
MDCの詳細については「*MDC*」を参照してください。

クラス名

ch.qos.logback.classic.sift.MDCBasedDiscriminator

プロパティ

プロパティ名	タイプ	必須設定	説明
key	String	○	分別を行うMDCのキーを指定します。
defaultValue	String	○	key で指定したMDCのキーが指す値が null であった場合に代替される文字列を指定します。

Filter

この設定はAppenderで設定可能です。<filter> で設定します。
Filterは、以下の役割を担います。

- ログ出力時に出力先に対してそのメッセージを出力するかの判定

この設定は <filter> を複数記述することで複数指定可能です。複数指定を行った場合、上から記述を行った順番に判定を行います。
以下は「*ConsoleAppender*」にFilterを指定した場合の im_logger.xml の例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
      <level>WARN</level>
      <onMatch>DENY</onMatch>
    </filter>
    <filter class="ch.qos.logback.classic.filter.LevelFilter">
      <level>INFO</level>
      <onMatch>DENY</onMatch>
    </filter>
    <encoder>
      <outputPatternAsHeader>true</outputPatternAsHeader>
      <pattern>[%level] %logger{10} - [%X{log.message.code}] %msg%n</pattern>
    </encoder>
  </appender>

</configuration>
```

この例では、以下のFilterが設定されています。

- ログレベルWARNであるログメッセージを拒否する
- ログレベルINFOであるログメッセージを拒否する

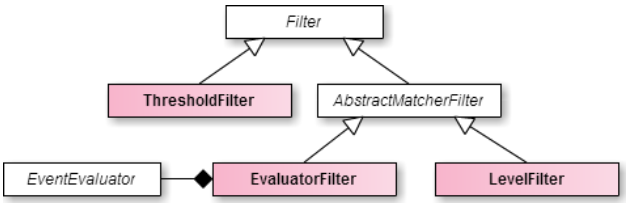
このAppenderを利用した場合、ログレベルERROR、DEBUG、TRACEであるログのみ出力されます。
「*SQLログを出力する*」では、Filterを利用した設定例を紹介していますので、併せて参照してください。

<filter> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	×	Filterの実装クラスの完全修飾クラス名を指定します。

Filterに対しての設定は <filter> の子要素で指定します。設定可能な値（プロパティ）は、Filterの実装により異なります。

Filterの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



以下では各種Filterの実装を紹介します。

> **コラム**
> 本書に記述されていないFilterについては「LogbackのWebサイト」を参照してください。

LevelFilter

ログレベルが特定の値と一致するかを判断し、一致／不一致の場合でのログ出力の有無をフィルタリングするFilterの実装です。

クラス名

ch.qos.logback.classic.filter.LevelFilter

プロパティ

プロパティ名	タイプ	必須設定	説明
level	LEVEL	○	一致／不一致を判断する「*ログレベル*」を指定します。
onMatch	FilterReply	×	ログメッセージのログレベルが <level> で指定したログレベルと一致している場合の、Filterの挙動を指定します。 指定可能な値は以下の通りです。 - DENY：出力先にログの出力を行いません。 - NEUTRAL：後続のFilterの判定に依存します。後続に設定されているFilterが無い場合はログの出力が行われます。 - ACCEPT：後続に設定されているFilterの判定を無視し、ログの出力を行います。 未指定の場合は、NEUTRAL を設定したものとして扱われます。
onMismatch	FilterReply	×	ログメッセージのログレベルが <level> で指定したログレベルと一致していない場合の、Filterの挙動を指定します。 指定可能な値と挙動は <onMatch> と同様です。 未指定の場合は、NEUTRAL を設定したものとして扱われます。

ThresholdFilter

ログレベルが特定の重要度であるかを判断し、特定の重要度を満たさないログメッセージの出力を抑制するFilterの実装です。

クラス名

ch.qos.logback.classic.filter.ThresholdFilter

プロパティ

プロパティ名	タイプ	必須設定	説明

level	LEVEL	○	重要度を判断する「*ログレベル*」を指定します。 例） INFO を指定した場合、ERROR、WARN、INFO 以外のログ出力を抑制します。

EvaluatorFilter

ログメッセージが特定の条件に、一致した場合／不一致の場合でのログ出力の有無をフィルタリングするFilterの実装です。
条件の判定は <evaluator> に依存します。

クラス名

ch.qos.logback.core.filter.EvaluatorFilter

プロパティ

プロパティ名	タイプ	必須設定	説明
evaluator	*EventEvaluator*	×	「*EventEvaluator*」を指定します。 <evaluator> の class 属性が未指定の場合は「*JaninoEventEvaluator*」を利用します。
onMatch	FilterReply	×	<evaluator> の条件と一致している場合の、Filterの挙動を指定します。 指定可能な値は以下の通りです。 ▪ DENY：出力先にログの出力を行いません。 ▪ NEUTRAL：後続のFilterの判定に依存します。後続に設定されているFilterが無い場合はログの出力が行われます。 ▪ ACCEPT：後続に設定されているFilterの判定を無視し、ログの出力を行います。 未指定の場合は、NEUTRAL を設定したものとして扱われます。
onMismatch	FilterReply	×	<evaluator> の条件と一致していない場合の、Filterの挙動を指定します。 指定可能な値と挙動は <onMatch> と同様です。 未指定の場合は、NEUTRAL を設定したものとして扱われます。

EventEvaluator

<evaluator> で設定します。
EventEvaluatorは、以下の役割を担います。

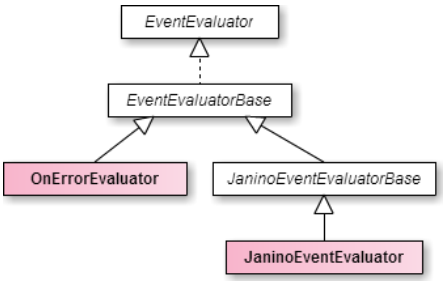
- ログ出力時の何かしらのイベントを引数とした条件の判定

<evaluator> で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	×	EventEvaluatorの実装クラスの完全修飾クラス名を指定します。

EventEvaluatorに対しての設定は <evaluator> の子要素で指定します。設定可能な値（プロパティ）は、EventEvaluatorの実装により異なります。

EventEvaluatorの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



以下では各種EventEvaluatorの実装を紹介します。

> **コラム**
> 本書に記述されていないEventEvaluatorについては「LogbackのWebサイト」を参照してください。

OnErrorEvaluator

ログメッセージのログレベルが ERROR であるかどうかを判定するEventEvaluatorの実装です。

クラス名

ch.qos.logback.classic.boolex.OnErrorEvaluator

プロパティ

設定可能なプロパティはありません。

JaninoEventEvaluator

評価式としてJavaの構文を指定し、一致するかどうか判定するEventEvaluatorの実装です。
このEventEvaluatorを利用するためには、「Janino ライブラリ」をクラスパス上に配置する必要があります。

> **注意**
>
> intra-mart Accel Platform ではJanino ライブラリは同梱されていません。
> このEventEvaluatorを利用するには別途ライブラリの配置作業が必要です。
> ライブラリのダウンロード、および、配置については「*SQLログを出力する*」を参照してください。

クラス名

`ch.qos.logback.classic.boolex.JaninoEventEvaluator`

プロパティ

プロパティ名	タイプ	必須設定	説明
expression	String	○	判定を行うための評価式を記述します。 評価式の詳細はLogbackのWebサイト「JaninoEventEvaluator」を参照してください。
matcher	Matcher	×	Matcherを指定します。 Matcherの詳細はLogbackのWebサイト「JaninoEventEvaluator」を参照してください。 `<matcher>` を複数記述することで複数指定可能です。

Encoder

この設定は一部のAppenderで設定可能です。`<encoder>` で設定します。
Encoderは、以下の役割を担います。

- ログ出力時の出力方法の決定
- ログ出力時のフォーマットの決定

以下は「*ConsoleAppender*」にEncoderを指定した場合の `im_logger.xml` の例です。

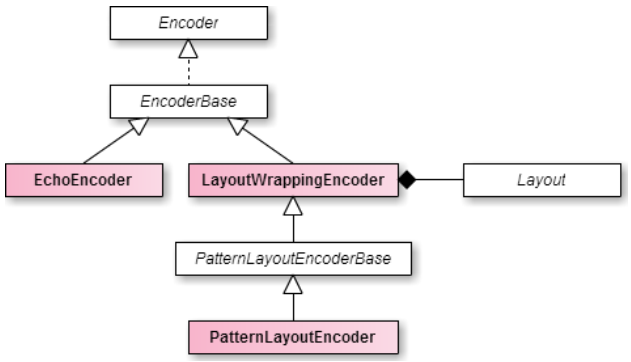
```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>

  <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <outputPatternAsHeader>true</outputPatternAsHeader>
      <pattern>[%level] %logger{10} - [%X{log.message.code}] %msg%n</pattern>
    </encoder>
  </appender>

</configuration>
```

`<encoder>` で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	×	Encoderの実装クラスの完全修飾クラス名を指定します。

Encoderに対しての設定は `<encoder>` の子要素で指定します。設定可能な値（プロパティ）は、Encoderの実装により異なります。

Encoderの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



以下では各種Encoderの実装を紹介します。

> **コラム**
> 本書に記述されていないEncoderについては「LogbackのWebサイト」を参照してください。

EchoEncoder

ログレベルとメッセージのみのログを出力するEncoderです。

クラス名

`ch.qos.logback.core.encoder.EchoEncoder`

プロパティ

設定可能なプロパティはありません。

PatternLayoutEncoder

様々な形式でログを出力することを可能としたEncoderです。指定したフォーマットパターンに従ってログが出力されます。

クラス名

`ch.qos.logback.classic.encoder.PatternLayoutEncoder`

プロパティ

プロパティ名	タイプ	必須設定	説明
pattern	String	○	出力時のフォーマットパターン。 フォーマットパターンで指定可能なパターン文字列については「*パターン文字列*」を参照してください。
outputPatternAsHeader	boolean	×	ログ出力開始時に、`<pattern>` に指定した文字列を出力するかを指定します。 `true` を指定した場合、Appenderの出力先に対してログメッセージを出力を開始する前に以下のような文字列が出力されます。 `#logback.classic pattern: [%level] %logger{10} - %msg%n` `false` を指定した場合は、出力を行いません。 未指定の場合は、`false` を設定したものとして扱われます。
charset	String	×	ログ出力に変換を行う文字コードを指定します。 未指定の場合は、文字コードの変換を行いません。
immediateFlush	boolean	×	バッファリングしている内容を即時書き込むかを指定します。 未指定の場合は、`true` を設定したものとして扱われます。 2019 Summer(Waltz) 以降でアップデートを行うと、system.log に `LogbackUtil` に関する Warning ログが出力されます。 warning ログの出力を回避するには、 logger.xml 内の `immediateFlush` タグを `encoder` タグの下から `appender` タグの下へ移動させてください。 2019 Spring(Violette) 以前のバージョンでは、Warning ログは出力されないため、`immediateFlush` タグの移動は不要です。 詳細は FAQ を確認ください。

LayoutWrappingEncoder

様々な形式でログを出力することを可能としたEncoderです。パターン文字列で指定したフォーマットに従ってログが出力されます。

クラス名

`ch.qos.logback.classic.encoder.LayoutWrappingEncoder`

プロパティ

プロパティ名	タイプ	必須設定	説明
layout	*Layout*	○	「*Layout*」を指定します。
charset	String	×	ログ出力に変換を行う文字コードを指定します。 未指定の場合は、文字コードの変換を行いません。
immediateFlush	boolean	×	バッファリングしている内容を即時書き込むかを指定します。 未指定の場合は、`true` を設定したものとして扱われます。 2019 Summer(Waltz) 以降でアップデートを行うと、system.log に `LogbackUtil` に関する Warning ログが出力されます。 warning ログの出力を回避するには、 logger.xml 内の `immediateFlush` タグを `encoder` タグの下から `appender` タグの下へ移動させてください。 2019 Spring(Violette) 以前のバージョンでは、Warning ログは出力されないため、`immediateFlush` タグの移動は不要です。 詳細は FAQ を確認ください。

Layout

この設定は一部のEncoderで設定可能です。`<layout>` で設定します。
Layoutは、以下の役割を担います。

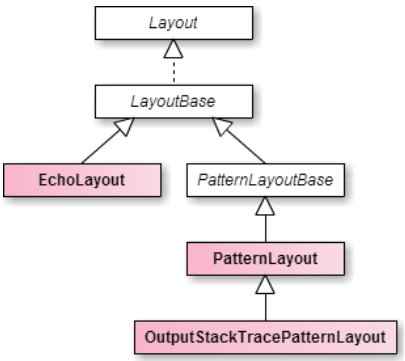
- ログ出力時のフォーマットの決定

`<layout>` で指定可能な属性設定は以下の通りです。

属性	必須設定	説明
class	×	Layoutの実装クラスの完全修飾クラス名を指定します。

Encoderに対しての設定は `<layout>` の子要素で指定します。設定可能な値（プロパティ）は、Layoutの実装により異なります。

Layoutの実装クラスとその他のクラスとの関連クラスの情報は以下の通りです。



以下では各種Layoutの実装を紹介します。

コラム
本書に記述されていないLayoutについては「LogbackのWebサイト」を参照してください。

EchoLayout

ログレベルとメッセージのみの文字列を生成するLayoutです。

クラス名

ch.qos.logback.core.layout.EchoLayout

プロパティ

設定可能なプロパティはありません。

PatternLayout

pattern に指定したパターン文字列のルールに従ってログに出力する文字列を生成します。

クラス名

ch.qos.logback.classic.PatternLayout

プロパティ

プロパティ名	タイプ	必須設定	説明
pattern	String	○	出力時のフォーマットパターン。 フォーマットパターンで指定可能なパターン文字列については「*パターン文字列*」を参照してください。
outputPatternAsHeader	boolean	×	ログ出力開始時に、<pattern> に指定した文字列を出力するかを指定します。 true を指定した場合、Appenderの出力先に対してログメッセージを出力を開始する前に以下のような文字列が出力されます。 `#logback.classic pattern: [%level] %logger{10} - %msg%n` false を指定した場合は、出力を行いません。 未指定の場合は、false を設定したものとして扱われます。

OutputStackTracePatternLayout

PatternLayoutを拡張したLayoutです。
このLayoutを利用することで通知された例外情報をExceptionログとして、別ファイルに出力します。

クラス名

jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout

プロパティ

プロパティ名	タイプ	必須設定	説明
pattern	String	○	出力時のフォーマットパターン。 このLayoutを利用すると、Exception情報が別ファイル（Exceptionログ）に出力されるため、通常はパターン文字列「*例外の抑止*」を含めたフォーマットパターンを指定します。 フォーマットパターンで指定可能なパターン文字列については「*パターン文字列*」を参照してください。
outputPatternAsHeader	boolean	×	ログ出力開始時に、<pattern> に指定した文字列を出力するかを指定します。 true を指定した場合、Appenderの出力先に対してログメッセージを出力を開始する前に以下のような文字列が出力されます。 `#logback.classic pattern: [%level] %logger{10} - %msg%n` false を指定した場合は、出力を行いません。 未指定の場合は、false を設定したものとして扱われます。
enableOutputStackTrace	boolean	×	Exceptionログの出力を行うかどうかを指定します。 未指定の場合は、true を設定したものとして扱われます。

stackTraceDir	String	×	Exceptionログの出力を行うディレクトリを指定します。 `<enableOutputStackTrace>` に `true` を指定する場合は必ず指定してください。
stackTraceFilename	String	×	Exceptionログの出力を行うファイル名のパターンを指定します。 パターンに指定した文字列の一部はファイル出力時に置換が行われます。置換が行われる文字列は以下の通りです。 ▪ `yyyy-MM-dd` のような `java.text.SimpleDateFormat` の形式をログ出力時の日時に置き換えます ▪ `%logId` を ログID に置き換えます。 `<enableOutputStackTrace>` に `true` を指定する場合は必ず指定してください。

コラム

ログID とは、intra-mart Accel Platform のLogger APIでログを出力する度に生成される一意な値です。MDCから取得可能です。MDCのキーは`log.id`です。
`<pattern>` に %X{log.id} を含めることで、Exceptionログとの紐付けができます。

システムログ

システムログは、ログ API を利用して出力される汎用的なログです。
具体的には、以下の内容がシステムログとして出力されます。（全て同一のログファイル system.log に出力されます。）

- サーバの運用状態を通知するためのログ
  - サーバの状態をトレースするために必要な情報が出力されます。
- エラー発生時のログ
  - サーバ運用中に発生した様々なエラーが出力されます。

モジュール	コアモジュール
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
 <jmxConfigurator />

 <!--
 <statusListener class="ch.qos.logback.core.status.OnConsoleStatusListener" />
 -->

 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
  <encoder>
   <outputPatternAsHeader>true</outputPatternAsHeader>
   <pattern>[%level] %logger{10} - [%X{log.message.code}] %msg%n</pattern>
   <!--
   <charset>Windows-31J</charset>
   -->
  </encoder>
 </appender>

 <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/system.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/system-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/system%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{255} %X{tenant.id} %X{log.id} %X{request.id} -
[%X{log.message.code}] %msg%nopex%n</pattern>

    <enableOutputStackTrace>true</enableOutputStackTrace>
    <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
    <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
   </layout>
  </encoder>
 </appender>

 <!--
 <appender name="EMAIL" class="ch.qos.logback.classic.net.SMTPAppender">
  <smtpHost>smtp_host</smtpHost>
  <to>to_mailaddress</to>
  <from>from_mailaddress</from>
  <layout class="ch.qos.logback.classic.PatternLayout">
   <Pattern>%date %-5level %logger{255} - %message%n</Pattern>
  </layout>
 </appender>
 -->

 <root>
  <level value="info" />
  <appender-ref ref="STDOUT" />
  <appender-ref ref="FILE" />
```

```
  </root>

  <!--
    - Logger for Axis2
   -->
  <logger name="org.apache.axis2.util.PrettyPrinter">
    <level value="error" />
  </logger>

  <!--
    - Logger for SpyDataSource
   -->
  <logger name="com.caucho.sql.spy">
    <level value="trace" />
  </logger>

  <!--
    - Logger for OpenPortal
   -->
  <logger name="debug.com.sun.portal.portletcontainer.impl">
    <level value="warn" />
  </logger>

  <logger name="com.sun.portal.portletcontainer.context.window.impl">
    <level value="warn" />
  </logger>

  <!--
    - Logger for WSRP
   -->
  <logger name="com.sun.xml.wss.logging.impl.filter">
    <level value="warn" />
  </logger>

  <!--
    - Logger for Session Timeout
   -->
  <logger name="error-page.session-timeout">
    <level value="off" />
  </logger>

</configuration>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	コンソール ファイル - ${im.log}/platform/system.log

出力パターン

利用可能なパターン文字列

システムログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

システムログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明

log.thread.group	×	スレッドグループ
log.id	○	ログ ID
request.id	○	リクエスト ID
log.message.code	○	ログメッセージコード
user.cd	×	ログ出力時のアカウントコンテキストのユーザコード intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
user.type	×	ログ出力時のアカウントコンテキストのユーザ種別 `administrator` : システム管理者 `platform` : ジョブなどのバックグラウンド `user` : 一般ユーザ intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
authenticated	×	ログ出力時のアカウントコンテキストの認証状態 `true` : 認証済み `false` : 未認証 intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。

コラム
アカウントコンテキストの詳細については、「アカウントコンテキストのJavaDoc」を参照してください。

注意

intra-mart Accel Platform 2013 Summer(Damask) 以前のバージョンを利用している場合、 初期状態のログ設定ファイルでは、system.log にメッセージコードが出力されません。

メッセージコードを出力するためには、`%CONTEXT_PATH%/WEB-INF/conf/log/im_logger.xml` に MDCキー「**log.message.code**」を設定してください。

特定用途ログ

特定用途ログとは、リクエスト情報、セキュリティ情報などの利用用途に特化したログの出力を行う複数のログを指します。
特定用途ログは、ログごとに専用の識別子が用意されており、ロガー名の先頭に必ず識別名が設定されます。
以下はリクエストログを出力するロガー名の例です。

```
REQUEST_LOG.jp.co.intra_mart.system.servlet.filter.RequestLogFilter
```

特定用途ログが利用する識別名は以下の通りです。

ログ	識別名
リクエストログ	REQUEST_LOG
画面遷移ログ	TRANSITION_LOG
セキュリティログ	SECURITY_LOG
マスタデータ更新ログ	MASTER_LOG
インポート・エクスポートログ	IMPORT_EXPORT_LOG
ユーザコンテキストログ	USER_CONTEXT_LOG
ネットワークログ	org.jgroups
EHCache ログ	net.sf.ehcache.util.UpdateChecker
IM-MessageHub ログ	MESSAGE_HUB_LOG
Hazelcastログ	jp.co.intra_mart.system.hazelcast
テーブルメンテナンス操作ログ	TABLE_EDIT
テーブルメンテナンスインポート・エクスポート 実行結果出力ログ	TABLE_EDIT.IMPORT_EXPORT
テーブルメンテナンスインポート・エクスポート 実行詳細出力ログ	TABLE_EDIT.IMPORT_EXPORT.FILE
IM-LogicDesigner ログ	LOGIC_FLOW_LOG
スマートメニューランキングログ	-

特定用途ログの一部のログでは `STDOUT` （ConsoleAppender）にて出力を行っています。`STDOUT` の設定は、「*システムログ*」で定義されているものを利用しています。

リクエストログ

リクエストログには、アプリケーションサーバのリクエスト・レスポンスに関連する情報が出力されます。
リクエストの処理が終了した際（＝レスポンスを返却する直前）に「INFO」レベルでログが出力され、リクエストを受け付けた時に「DEBUG」レベルでログが出力されます。
上記２つのタイミングで出力されるログを比較することにより、処理中の状態、つまり、リクエストを受け付けたが、レスポンスはまだ返却されていない状態であるリクエストを特定することが可能です。「リクエストを受け付けた時のログ」と「レスポンスを返却した時のログ」は、リクエスト ID で紐付けることが可能です。

> **注意**
> リクエストを受け付けたときのログを出力する場合はログレベルをDEBUG以下の重要度に設定する必要があります。

モジュール	コアモジュール
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_request.xml

```
<included>

 <!--
   - REQUEST_LOG
  -->
 <appender name="REQUEST_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/request.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/request-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/request%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %X{log.report.sequence} %-5level %logger{255} %X{tenant.id} %X{log.id} -
%X{client.session.id} %X{request.remote.host} %X{request.method} %X{request.url} %X{request.query_string} %X{request.url.referer}
%X{request.page.time} %X{request.accept.time} %X{request.id}%nopex%n</pattern>

    <enableOutputStackTrace>true</enableOutputStackTrace>
    <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
    <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
   </layout>
  </encoder>
 </appender>

 <logger name="REQUEST_LOG" additivity="false">
  <level value="info" />
  <appender-ref ref="REQUEST_FILE" />
 </logger>

</included>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	コンソール ファイル - ${im.log}/platform/request.log

出力パターン

利用可能なパターン文字列

リクエストログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	×	*ログメッセージ* リクエストを受け付けた時は IN レスポンスを返却した時は OUT が出力されます。
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

リクエストログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
log.thread.group	×	スレッドグループ
log.report.sequence	○	ログ出力順序番号
log.id	○	ログ ID
client.session.id	○	セッション ID
request.remote.host	○	リモートホスト intra-mart Accel Platform 2015 Winter(Lydia) で出力方式が変更されました。 詳しくは、後述のコラムを参照してください。
request.remote.address	×	リモートアドレス intra-mart Accel Platform 2015 Winter(Lydia) で出力方式が変更されました。 詳しくは、後述のコラムを参照してください。
request.method	○	HTTPメソッド
request.url	○	リクエストURL
request.query_string	○	クエリ文字列
request.url.referer	○	リクエスト発生元URL
request.page.time	○	ページ処理時間。ミリ秒で出力されます。
request.accept.time	○	リクエストの受付時間。 [yyyy-MM-dd HH:mm:ss,SSS] 形式で出力されます。
request.id	○	リクエスト ID
user.cd	×	ログ出力時のアカウントコンテキストのユーザコード intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
user.type	×	ログ出力時のアカウントコンテキストのユーザ種別 `administrator` : システム管理者 `platform` : ジョブなどのバックグラウンド `user` : 一般ユーザ intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
authenticated	×	ログ出力時のアカウントコンテキストの認証状態 `true` : 認証済み `false` : 未認証 intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。

コラム
アカウントコンテキストの詳細については、「アカウントコンテキストのJavaDoc」を参照してください。

注意
リモートアドレス（request.remote.address）、および、リモートホスト（request.remote.host） の出力方式が変更されました。

- intra-mart Accel Platform 2015 Winter(Lydia) 以降の場合

  リモートアドレス、リモートホスト 共に、 InetAddressDetector#getRemoteAddress() で取得した java.net.InetAddress の #getHostAddress() の 値が設定されます。

  InetAddressDetector は、IPアドレス取得元設定 に従い、リクエストからリモートホストのIPアドレスを取得します。

  なお、リモートホストには **IPアドレス** が出力されます。（パフォーマンス向上のためにホスト名解決を行いません。）
  ホスト名解決を行いたい場合は、以下のサービス定義ファイルを追加してください。

  - %CONTEXT_PATH%/WEB-INF/classes/META-INF/services/jp.co.intra_mart.system.http.address.resolver.RemoteAddressResolverDelegate

    ```
    jp.co.intra_mart.system.http.address.resolver.impl.HostNameConsciousRemoteAddressResolverByInetAddressDetector
    ```

- intra-mart Accel Platform 2015 Summer(Karen) 以前の場合

  リモートアドレス ： HttpServletRequest#getRemoteAddr() の 値。
  リモートホスト　 ： HttpServletRequest#getRemoteHost() の 値。

画面遷移ログ

画面遷移ログには、利用者の操作により行われる画面遷移の情報が出力されます。

モジュール	コアモジュール
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_transition.xml

```
<included>

 <!--
   - TRANSITION_LOG
  -->
 <appender name="TRANSITION_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/transition.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/transition-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/transition%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %X{log.report.sequence} %-5level %logger{255} %X{tenant.id} %X{log.id} -
%X{transition.log.type.id} %X{request.remote.address} %X{request.remote.host} %X{transition.access.user.id} %X{client.session.id}
%X{transition.path.page.next} %X{transition.time.response} %X{transition.exception.name} %X{transition.exception.message}
%X{transition.path.page.previous} %X{request.id}%nopex%n</pattern>

    <enableOutputStackTrace>true</enableOutputStackTrace>
    <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
    <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
   </layout>
  </encoder>
 </appender>

 <logger name="TRANSITION_LOG" additivity="false">
  <level value="info" />
  <appender-ref ref="TRANSITION_FILE" />
 </logger>

</included>
```

遷移タイプによる挙動の違いについて

MDCにより取得可能な遷移元画面のパス、および、遷移先画面のパスは以下のように出力されます。

遷移タイプ	遷移元画面のパス	遷移先画面のパス
REQUEST	HTTP ヘッダ「Referer」のサーブレットパス部分（HTTP ヘッダ「Referer」が取得できない場合は出力されません）	URL のサーブレットパス部分
FORWARD	RequestDispatcher#forward() を実行する前の遷移先画面パス	ServletRequest#getRequestDispatcher() 時に指定したパス
INCLUDE	RequestDispatcher#include() を実行する前の遷移先画面パス	ServletRequest#getRequestDispatcher() 時に指定したパス

いずれの遷移タイプでも、遷移元画面、および、遷移先画面がスクリプト開発モデルである場合には、スクリプト開発モデルの画面のパスが出力されます。

例えば、3 つの JSP 「1_request.jsp」、「2_forwarded.jsp」、「3_included.jsp」が存在し、「1_request.jsp」 から 「2_forwarded.jsp」へforwardし、forward先の「2_forwarded.jsp」 内で 「3_included.jsp」 をincludeする場合、遷移元画面、および、遷移先画面のパスは以下のように出力されます。
（なお、この時のリクエスト ID は、同一の ID が出力されます。）

出力順	遷移タイプ	遷移元画面のパス	遷移先画面のパス
1	REQUEST	▪	1_request.jsp
2	FORWARD	1_request.jsp	2_forwarded.jsp
3	INCLUDE	2_forwarded.jsp	3_included.jsp

応答時間は、遷移タイプによって出力される値が異なります。

REQUEST	リクエストの処理開始から、レスポンスを返却するまでの時間をあらわします。
FORWARD	リクエストの処理開始から、FORWARD した時点 （＝ディスパッチ前の時点）までの時間をあらわします。
INCLUDE	リクエストの処理開始から、INCLUDE した時点（＝ディスパッチ前の時点）までの時間をあらわします。

> **注意**
>
> 遷移タイプ FORWARD、および、INCLUDE のログは、ServletRequest#getRequestDispatcher() で取得した RequestDispatcher を利用して forward/includeされた場合に出力されます。（ServletContext 経由で RequestDispatcher を取得した場合は出力されません）

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	コンソール ファイル - ${im.log}/platform/transition.log

出力パターン

利用可能なパターン文字列

画面遷移ログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

画面遷移ログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
log.thread.group	×	スレッドグループ
log.report.sequence	○	ログ出力順序番号
log.id	○	ログ ID
client.session.id	○	セッション ID
transition.log.type.id	○	遷移タイプ REQUEST : 通常のリクエスト FORWARD : javax.servlet.RequestDispatcher#forward() 時の遷移 INCLUDE : javax.servlet.RequestDispatcher#include() 時の遷移
transition.access.user.id	○	ユーザコード
transition.path.page.next	○	遷移先画面のパス

transition.path.page.previous	○	遷移元画面のパス
transition.time.response	○	応答時間
transition.exception.name	○	例外名
transition.exception.message	○	例外メッセージ
request.remote.host	○	リモートホスト intra-mart Accel Platform 2015 Winter(Lydia) で出力方式が変更されました。 詳しくは「 *リクエストログ* 」の「 *利用可能なMDCキー* 」を参照してください。
request.remote.address	○	リモートアドレス intra-mart Accel Platform 2015 Winter(Lydia) で出力方式が変更されました。 詳しくは「 *リクエストログ* 」の「 *利用可能なMDCキー* 」を参照してください。
request.id	○	リクエスト ID
user.cd	×	ログ出力時のアカウントコンテキストのユーザコード intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
user.type	×	ログ出力時のアカウントコンテキストのユーザ種別 administrator : システム管理者 platform : ジョブなどのバックグラウンド user : 一般ユーザ intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
authenticated	×	ログ出力時のアカウントコンテキストの認証状態 true : 認証済み false : 未認証 intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
client.type	×	ログ出力時のクライアントコンテキストのクライアントタイプ pc : PC sp : スマートフォン intra-mart Accel Platform 2017 Winter(Rebecca) 以降、利用可能です。

コラム
アカウントコンテキストの詳細については、「アカウントコンテキストのJavaDoc」を参照してください。

コラム
クライアントコンテキストの詳細については、「ClinetContextコンテキストのJavaDoc」を参照してください。

セキュリティログ

セキュリティログには、認証・認可の結果などのセキュリティに関連する情報が出力されます。
このログとリクエストログを合わせて解析することにより、不正アクセスを見つけ出すための情報を得ることも可能です。

モジュール	コアモジュール
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_security.xml

```
<included>

 <!--
   - SECURITY_LOG
  -->
 <appender name="SECURITY_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/security.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/security-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/security%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %X{log.report.sequence} %-5level %logger{255} %X{tenant.id} %X{log.id} -
%X{security.id.session} %X{security.id.account} %X{security.id.usertype} [%X{log.message.code}] %msg
%X{request.id}%nopex%n</pattern>

    <enableOutputStackTrace>true</enableOutputStackTrace>
    <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
    <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
   </layout>
  </encoder>
 </appender>

 <logger name="SECURITY_LOG" additivity="false">
  <level value="warn" />
  <appender-ref ref="STDOUT" />
  <appender-ref ref="SECURITY_FILE" />
 </logger>

</included>
```

ログレベルごとの出力内容

ログレベルごとのセキュリティログの出力内容を紹介します。

WARNレベル

このログレベルのセキュリティログが短時間で連続して出力されている場合、サーバに対して何らかの攻撃が行われている可能性があります。
WARNレベルにて出力される内容は以下の通りです。

ケース	メッセージ例
認証に失敗した場合	ログインに失敗しました。ログインパスワードが間違っています。
認証確認に失敗した場合	認証確認に失敗しました。
権限の無いページへのアクセスした場合	アクセスする権限がありません。
SecureToken機能にてセキュアトークンが不正である場合	セキュアトークンが不正です。
アプリケーションライセンスを保持していない利用者がライセンスが必要な機能にアクセスした場合	アプリケーション「（プロダクトID）」のライセンスが設定されていないため、アクセスできません。
2段階認証の検証に失敗した場合	2段階認証に失敗しました。
管理者がユーザの2段階認証を必須にしており、ユーザのログインと同時に行う2段階認証設定の有効化に失敗した場合	2段階認証の設定に失敗しました。
管理者がユーザの2段階認証を必須にしており、ユーザのログインと同時に行う2段階認証設定の有効化のための確認コードの検証に失敗した場合	多要素認証の設定に失敗しました。

INFOレベル

INFOレベルにて出力される内容は以下の通りです。

ケース	メッセージ例
認証に成功した場合	ログインに成功しました。
認証確認に成功した場合	認証確認に成功しました。

DEBUGレベル

DEBUGレベルにて出力される内容は以下の通りです。

ケース	メッセージ例
ページへのアクセスに成功（権限があるページにアクセス）した場合	アクセスに成功しました。
SecureToken機能にてセキュアトークンのチェックに問題が無い場合	セキュアトークンのチェックに成功しました。
アプリケーションライセンスを保持している利用者がライセンスが必要な機能にアクセスした場合	アプリケーション「（プロダクトID）」のライセンスチェックに成功しました。
2段階認証設定が行われているユーザに２段階認証を要求した場合	ユーザに2段階認証を要求します。
管理者がユーザの2段階認証を必須にしており、ユーザのログインと同時に行う2段階認証設定を要求した場合	ユーザに2段階認証の設定を要求します。

標準出力設定

ログレベル（初期値）	WARN
出力先（初期値）	コンソール ファイル - `${im.log}/platform/security.log`

出力パターン

利用可能なパターン文字列

セキュリティログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

セキュリティログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
log.thread.group	×	スレッドグループ
log.report.sequence	○	ログ出力順序番号
log.id	○	ログ ID
security.id.session	○	セッション ID
security.id.account	○	セキュリティに関する操作を行ったユーザコード 例として、ログイン画面でユーザコードに `login_user` と入力し、ログインに失敗した場合は、 `login_user` が取得されます。
security.id.usertype	○	セキュリティに関する操作を行った際の、操作に関わるユーザ種別 `administrator` : システム管理者 `platform` : ジョブなどのバックグラウンド `user` : 一般ユーザ 例として、システム管理者のログインに失敗した場合は、`administrator` が取得されます。
log.message.code	○	ログメッセージコード
request.id	○	リクエスト ID
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。

コラム
アカウントコンテキストの詳細については、「アカウントコンテキストのJavaDoc」を参照してください。

マスタデータ更新ログ

マスタデータ更新ログには、intra-mart Accel Platform で保持するマスタ情報に対して、更新処理（登録、更新、削除）に関する情報が出力されます。
また、トランザクションの開始、終了も出力し、トランザクション単位でのマスタ情報の更新結果を記録します。ログはすべてログレベルINFOで出力されます。

出力対象は、インポート・エクスポート、テナント管理、ジョブスケジューラ、ポータルモジュールの更新系のAPIのみです。

マスタデータ更新ログで出力するメッセージはすべてプロパティファイルから取得しています。メッセージが定義されているプロパティファイルの配置場所や、各APIが出力するメッセージの内容については、別紙「マスタデータ更新ログメッセージ一覧」を参照してください。

モジュール	コアモジュール
設定場所	`%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_update_master_data.xml`

```
<included>

 <!--
   - MASTER_LOG
  -->
 <appender name="MASTER_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/report/update_master_data.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/report/update_master_data-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/report/update_master_data%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %logger{255} %X{tenant.id} %X{log.id} %X{request.id} %X{client.session.id}
%X{masterlog.user.cd} %X{request.remote.address} %X{masterlog.transaction.id} %X{masterlog.function.type} [%X{log.message.code}]
%message %X{masterlog.result}%nopex%n</pattern>
    <enableOutputStackTrace>true</enableOutputStackTrace>
    <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
    <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
   </layout>
  </encoder>
 </appender>

 <logger name="MASTER_LOG" additivity="false">
  <level value="info" />
  <appender-ref ref="MASTER_FILE" />
 </logger>

</included>
```

制限事項

別紙「マスタデータ更新ログメッセージ一覧」に定義されていないAPIでマスタデータにアクセスした場合は、ログ出力の対象外です。
別紙「マスタデータ更新ログメッセージ一覧」に定義されていないAPI以外でマスタデータ更新ログの出力を行いたい場合は、
`jp.co.intra_mart.system.log.masterlog.MasterLog` の `log()` メソッドを利用してログ出力を行ってください。

トランザクション開始・終了ログは、intra-mart標準のトランザクション管理を利用している場合のみ出力されます。
intra-mart Accel Platform 標準のトランザクション管理機能を利用していない場合以下の問題が発生します。

- トランザクション開始・終了ログは出力されません。
- トランザクションIDが出力されません。

intra-mart Accel Platform 標準のトランザクション管理機能を利用せずにマスタデータ更新ログでトランザクションに関連する情報を出力するには
`jp.co.intra_mart.system.log.masterlog.MasterLog` の `logBeginTransaction()` メソッドまたは `logFinishTransaction()` メソッドを利用してトランザクションログの出力を行ってください。

内部統制対応

このログは、いつ、誰が、どこから、どのような操作を行ったか、その結果はどうなったのかを記憶することにより、内部統制における「モニタリング」の有効性を確保するための手段として利用が可能です。ただし、この機能では内部統制対応として必要最低限の情報のみ出力されるため、「*制限事項*」に記述されている制限が存在します。システムの監査基準に照らし合わせて足りない項目や処理については、ログ設定ファイルで必要な項目を出力するように設定する、または、「*カスタマイズ*」を行う必要があります。
ログの解析については「*ログを解析する*」を参照してください。

カスタマイズ

マスタデータ更新ログ機能では、`jp.co.intra_mart.system.log.masterlog.MasterLog` を使用してログの出力を行います。
このクラスでは、ログ出力時に必要なパラメータをMDCにセットし、引数のメッセージキーからメッセージを取得してログ出力を行います。
トランザクション管理処理では、トランザクション IDの生成やMDCへの登録・削除を行います。
詳細は `jp.co.intra_mart.system.log.masterlog.MasterLog` のAPIリストを参照してください。

トランザクション ID

トランザクションとAPIからのマスタ情報更新結果を結びつけるためのユニークなIDです。
実際にトランザクションが保持しているIDではなく、マスタデータ更新ログ専用のIDです。
マスタ情報更新処理のトランザクション IDが同一である場合、それらの処理は同一のトランザクションで行われています。
以下はトランザクション管理により、アカウント aoyagi に対しての複数の更新処理がロールバックされた場合のマスタデータ更新ログの出力結果です。（トランザクション ID、処理区分、メッセージ、更新結果を出力しています。）

```
5i4dh906hbb2m5c TB トランザクションを開始しました
5i4dh906hbb2m5c U  アカウント(aoyagi)を更新します。 成功
5i4dh906hbb2m5c U  アカウント(aoyagi)を更新します。 成功
5i4dh906hbb2m5c U  アカウント(aoyagi)を更新します。 失敗
5i4dh906hbb2m5c TR トランザクションをロールバックしました
```

ロガー名について

マスタデータ更新ログでは、ロガー名を以下のように定義しています。

- 「識別名」 ＋ 「APIのクラス名」

例）
AccountInfoManagerの場合

```
MASTER_LOG.jp.co.intra_mart.foundation.admin.account.AccountInfoManager
```

設定ファイルに個別にロガーを登録することで、APIごとにログの出力を抑制することが可能です。
AccountInfoManagerの更新結果を出力しない場合、以下を設定ファイルに追記します。

```
<logger name="MASTER_LOG.jp.co.intra_mart.foundation.admin.account.AccountInfoManager" additivity="false">
  <level value="off" />
</logger>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	コンソール ファイル - ${im.log}/platform/report/update_master_data.log

出力パターン

利用可能なパターン文字列

マスタデータ更新ログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

マスタデータ更新ログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
log.id	○	ログ ID
client.session.id	○	セッション ID
masterlog.user.cd	○	ユーザコード
masterlog.transaction.id	○	トランザクション ID

masterlog.function.type	○	処理区分 C：生成 U：更新 D：削除 TB：トランザクションの開始 TC：トランザクションのコミット TR：トランザクションのロールバック
masterlog.result	○	更新結果 更新API単位での処理の成功／失敗を出力します。
request.id	○	リクエスト ID
request.remote.address	○	リモートアドレス
log.message.code	○	ログメッセージコード
user.cd	×	ログ出力時のアカウントコンテキストのユーザコード intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
user.type	×	ログ出力時のアカウントコンテキストのユーザ種別 administrator：システム管理者 platform：ジョブなどのバックグラウンド user：一般ユーザ intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
authenticated	×	ログ出力時のアカウントコンテキストの認証状態 true：認証済み false：未認証 intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。

コラム
アカウントコンテキストの詳細については、「アカウントコンテキストのJavaDoc」を参照してください。

インポート・エクスポートログ

インポート・エクスポートログには intra-mart Accel Platform が提供するインポータ／エクスポータの実行に関する情報が出力されます。

インポート・エクスポート処理が開始・正常終了した際は、ログレベル INFO でログが出力されます。
インポート・エクスポート処理に失敗したが、後続のデータの処理を続行した際は、ログレベル WARN でログが出力されます。
インポート・エクスポート処理が異常終了した際は、ログレベル ERROR でログが出力されます。

モジュール	インポート・エクスポート
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_import_export.xml

```
<included>

  <!--
   - IMPORT_EXPORT_LOG
   -->
  <appender name="IMPORT_EXPORT_FILE" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
      <key>data.io.execute.id</key>
      <defaultValue>unknown</defaultValue>
    </discriminator>
    <timeout>10 seconds</timeout>
    <sift>
      <appender name="FILE-${data.io.execute.id}" class="jp.co.intra_mart.common.platform.log.appender.SystemStorageAppender">
        <filter class="jp.co.intra_mart.common.platform.log.filter.BuildAppenderFilter"/>
        <file>log/import-export/${data.io.execute.id}.log</file>
        <append>true</append>
        <encoder>
          <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] %X{tenant.id} %X{log.id} : [%-5level] %X{data.io.record.num} -
[%X{log.message.code}] %msg : %X{data.io.data.key}%n%ex</pattern>
        </encoder>
      </appender>
    </sift>
  </appender>

  <logger name="IMPORT_EXPORT_LOG" additivity="false">
    <level value="info"/>
    <appender-ref ref="STDOUT"/>
    <appender-ref ref="IMPORT_EXPORT_FILE"/>
  </logger>

</included>
```

出力を行っている実装

intra-mart Accel Platform が提供しているインポート・エクスポートログの出力を行う機能は以下の通りです。

- テナント管理機能
  - カレンダーインポート
  - 日付情報セットインポート
  - 日付情報インポート
  - カレンダーマージ設定インポート
  - アカウントインポート
  - ロールインポート
  - 認可（リソースグループ）インポート
  - 認可（リソース）インポート
  - 認可（サブジェクトグループ）インポート
  - 認可（ポリシー）インポート
  - メニューグループカテゴリインポート
  - メニューグループインポート
  - カレンダーエクスポート
  - 日付情報セットエクスポート
  - 日付情報エクスポート
  - カレンダーマージ設定エクスポート
  - アカウントエクスポート
  - ロールエクスポート
  - 認可（リソースグループ）エクスポート
  - 認可（リソース）エクスポート
  - 認可（サブジェクトグループ）エクスポート
  - 認可（ポリシー）エクスポート
  - メニューグループカテゴリエクスポート
  - メニューグループエクスポート
- ジョブスケジューラ
  - ジョブインポート
  - ジョブエクスポート
- インポート・エクスポート
  - 拡張データインポート

標準出力設定

ログレベル（初期値） INFO

出力先（初期値）	コンソール システムストレージ - log/import-export/${data.io.execute.id}

コラム

${data.io.execute.id} は <discriminator> 設定で割り当てられた実行 IDが取得可能なプロパティです。
プロパティについては「*プロパティ*」を参照してください。

出力パターン

利用可能なパターン文字列

インポート・エクスポートログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

インポート・エクスポートログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
data.io.record.num	○	処理番号
data.io.data.key	○	データキー 対象となるデータを特定するためのキー
data.io.execute.id	×	実行 ID
log.message.code	○	ログメッセージコード
user.cd	×	ログ出力時のアカウントコンテキストのユーザコード intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
user.type	×	ログ出力時のアカウントコンテキストのユーザ種別 administrator：システム管理者 platform：ジョブなどのバックグラウンド user：一般ユーザ intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。
authenticated	×	ログ出力時のアカウントコンテキストの認証状態 true：認証済み false：未認証 intra-mart Accel Platform 2014 Spring(Granada) 以降、利用可能です。

コラム

アカウントコンテキストの詳細については、「アカウントコンテキストのJavaDoc」を参照してください。

ユーザコンテキストログ

ユーザコンテキストログには、ユーザコンテキスト構築処理の所要時間が出力されます。

注意

- 開発時用のログであるため、インストール時にはログレベルが「OFF」に設定されています。
  必要に応じてログレベルを変更してください。
- ユーザコンテキストログを出力する場合には、ログレベルを「DEBUG」レベルに設定する必要があります。

モジュール	ユーザコンテキスト
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_user_context.xml

```
<included>

  <!--
    - USER_CONTEXT_LOG
   -->
  <appender name="USER_CONTEXT" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${im.log}/platform/user_context.log</file>
    <append>true</append>

    <!--
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>
        ${im.log}/platform/user_context-%d{yyyy-MM-dd}.log
      </fileNamePattern>
    </rollingPolicy>
    -->

    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
      <fileNamePattern>${im.log}/platform/user_context%i.log</fileNamePattern>
      <minIndex>1</minIndex>
      <maxIndex>5</maxIndex>
    </rollingPolicy>

    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <maxFileSize>10MB</maxFileSize>
    </triggeringPolicy>

    <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
      <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
        <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}]  [%thread]  %X{log.report.sequence}  %-5level  %X{tenant.id}  %X{log.id}
%X{user_context_execution}  %X{user_context_user_cd}  -  %msg%nopex%n</pattern>

        <enableOutputStackTrace>true</enableOutputStackTrace>
        <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
        <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
      </layout>
    </encoder>
  </appender>

  <logger name="USER_CONTEXT_LOG" additivity="false">
    <level value="off" />
    <appender-ref ref="USER_CONTEXT" />
  </logger>

</included>
```

ログレベル（初期値）	OFF
出力先（初期値）	ファイル - ${im.log}/platform/user_context.log

出力パターン

利用可能なパターン文字列

ユーザコンテキストログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

ユーザコンテキストログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
log.report.sequence	○	ログ出力順序番号
log.id	○	ログ ID
user_context_user_cd	○	ユーザコード
user_context_execution	○	ユーザコンテキストに値をセットするためのSQL実行時間
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID

ネットワークログ

ネットワークログには、intra-mart Accel Platform のサービス間で行われるネットワーク通信に関する情報が出力されます。

> **注意**
> このログではログ IDなどの値をMDCで取得することはできません。

モジュール	コアモジュール
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_network.xml

```
<included>

 <!--
   - NETWORK_LOG
  -->
 <appender name="NETWORK_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/network.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/network-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/network%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder>
   <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{255} - %msg%n</pattern>
  </encoder>
 </appender>

 <!-- Formerly Known As Logger Name "NETWORK_LOG" -->
 <logger name="org.jgroups" additivity="false">
  <level value="off" />
  <appender-ref ref="STDOUT" />
  <appender-ref ref="NETWORK_FILE" />
 </logger>

</included>
```

標準出力設定

ログレベル（初期値）	OFF
出力先（初期値）	コンソール ファイル - ${im.log}/platform/network.log

出力パターン

利用可能なパターン文字列

リクエストログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*

EHCache ログ

EHCache ログには、 intra-mart Accel Platform が内包する EHCache のバージョンチェックに関する情報が出力されます。
外部とのネットワーク通信が可能な状態、かつ EHCache のバージョンが最新ではない場合にログが出力されます。

注意

- EHCache のバージョンチェック機能は intra-mart Accel Platform 2015 Summer(Karen) で廃止されました。
- intra-mart Accel Platform 2015 Spring(Juno) 以前で EHCache ログを出力する場合は、ログレベルを「INFO」以上に設定してください。 または、 im_logger_ehcache.xml を削除してください。

モジュール	サービス機構モジュール
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_ehcache.xml

```
<included>
  <logger name="net.sf.ehcache.util.UpdateChecker" additivity="false">
    <level value="off" />
  </logger>
</included>
```

標準出力設定

ログレベル（初期値）	OFF

IM-MessageHub ログ

IM-MessageHub ログには、IM-MessageHub を利用した配信機能の配信状況が出力されます。
IM-MessageHub とは、ユーザへの通知を目的としたメッセージの配信機能のフレームワークです。
IM-MessageHub を利用することで、アプリケーションは、メール、IMBox の ApplicationBox、IM-Notice 等の様々な配信先メディアへの通知処理を統一的に扱うことが可能になります。

IM-MessageHub ログでは、IM-MessageHub を利用した場合の、配信先メディア、送信者、宛先、配信結果等の情報が出力され、通知先への配信が実行されたことを確認することができます。

配信処理が開始・正常終了した際は、ログレベル INFO でログが出力されます。
配信処理に失敗したが、後続の宛先への配信処理を続行した際は、ログレベル WARN でログが出力されます。
配信処理が異常終了した際は、ログレベル ERROR でログが出力されます。

> **注意**
> IM-MessageHub ログは、intra-mart Accel Platform 2014 Winter(Iceberg) 以降で利用可能です。

モジュール	IM-MessageHub
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_message_hub.xml

```
<included>

 <!--
   - MESSAGE_HUB_LOG
  -->
 <appender name="MESSAGE_HUB_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/message_hub.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/message_hub-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/message_hub%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.common.platform.log.layout.OutputStackTracePatternLayout">
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %X{log.report.sequence} %-5level %logger{255} %X{tenant.id} %X{log.id} -
%X{message_hub.message.id} %X{message_hub.event.id} [%X{log.message.code}] %msg %X{request.id}%nopex%n</pattern>

    <enableOutputStackTrace>true</enableOutputStackTrace>
    <stackTraceDir>${im.log}/platform/exception/</stackTraceDir>
    <stackTraceFilename>'exception_'yyyy-MM-dd_HH-mm-ss'_%logId.log'</stackTraceFilename>
   </layout>
  </encoder>
 </appender>

 <logger name="MESSAGE_HUB_LOG" additivity="false">
  <level value="info" />
  <appender-ref ref="MESSAGE_HUB_FILE" />
 </logger>

</included>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	ファイル - ${im.log}/platform/message_hub.log

出力パターン

利用可能なパターン文字列

IM-MessageHub ログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

IM-MessageHub ログで利用可能なMDCのキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
log.thread.group	×	スレッドグループ
log.id	○	ログ ID
message_hub.message.id	○	メッセージ ID
message_hub.event.id	○	イベント ID メッセージの配信を行ったアプリケーション、および、処理を特定するユニークな文字列です。
log.message.code	○	ログメッセージコード
request.id	○	リクエスト ID
user.cd	×	ログ出力時のアカウントコンテキストのユーザコード
user.type	×	ログ出力時のアカウントコンテキストのユーザ種別 `administrator` : システム管理者 `platform` : ジョブなどのバックグラウンド `user` : 一般ユーザ
tenant.id	○	ログ出力時のアカウントコンテキストのテナント ID
authenticated	×	ログ出力時のアカウントコンテキストの認証状態 `true` : 認証済み `false` : 未認証

Hazelcastログ

Hazelcastログには、セッション管理 組み込みHazelcast連携のセッション情報管理に関するログが出力されます。

> **注意**
> Hazelcastログは、intra-mart Accel Platform 2017 Spring(Portland) 以降で利用可能です。

モジュール	セッション管理 組み込みHazelcast連携
設定場所	`%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_hazelcast.xml`

```
<included>

 <!--
   - HAZELCAST_LOG
  -->
 <appender name="HAZELCAST_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/platform/hazelcast.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/hazelcast-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/hazelcast%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

          <encoder>
   <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{255} - %msg%n</pattern>
  </encoder>
 </appender>

 <logger name="jp.co.intra_mart.system.hazelcast" additivity="false">
  <level value="info" />
  <appender-ref ref="HAZELCAST_FILE" />
 </logger>
 <logger name="jp.co.intra_mart.system.hazelcast.spi.impl.servicemanager.impl.ServiceManagerImpl" additivity="false">
  <level value="error" />
  <appender-ref ref="HAZELCAST_FILE" />
 </logger>

</included>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	ファイル - ${im.log}/platform/hazelcast.log

出力パターン

利用可能なパターン文字列

Hazelcastログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*

テーブルメンテナンス操作ログ

テーブルメンテナンス操作ログは、テーブルメンテナンスのレコード編集画面において
どのテーブルに対してどのような操作を行ったかをSQL形式にて出力します。
またこのログは、編集処理に成功した場合にのみ、出力されます。

> **注意**
>
> テーブルメンテナンス操作ログは、intra-mart Accel Platform 2015 Summer(Karen) 以降で利用可能です。

モジュール	TableMaintenance
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_tablemaintenance.xml

```
<included>

 <appender name="TABLE_EDIT" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/product/tablemaintenance/tablemaintenance.log</file>
  <append>true</append>

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/product/tablemaintenance/tablemaintenance%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="ch.qos.logback.classic.PatternLayout">
    <!--
     vctm.connect_id: データベース接続ID
     vctm.tablename: 対象テーブル名
     vctm.statement_type: 行われた操作の種類。下記のいずれかです。
       ・データの表示(SELECT)
       ・データの挿入(INSERT)
       ・データの更新(UPDATE)
       ・データの削除(DELETE)
     vctm.tenant_database: true:テナントデータベース　false:シェアドデータベース
     vctm.sql_result: SQLの実行結果
       ・INSERT, UPDATE, DELETEの場合
        処理件数
       ・SELECTの場合
        取得されたレコードの内容　※出力されるデータ量が膨大になる可能性があります

     ログメッセージ本体(message)
      発行されたSQLとそのとき利用されたパラメータデータが出力されます。
    -->
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] [%X{user.cd}] [%X{vctm.connect_id}] [%X{vctm.tablename}]
[%X{vctm.statement_type}] %message%n</pattern>
   </layout>
  </encoder>
 </appender>

 <!--
  レコード編集画面における下記の操作に対するログの出力を行います。
   ・データの閲覧(SELECT)
   ・データの挿入(INSERT)
   ・データの更新(UPDATE)
   ・データの削除(DELETE)
  上記の操作ログは、infoレベルで出力されます。

  ロガー名： TABLE_EDIT.{データベース接続ID}.{テーブル名}
 -->
 <logger name="TABLE_EDIT" additivity="false">
  <level value="info" />
  <appender-ref ref="TABLE_EDIT" />
 </logger>

</included>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	ファイル - ${im.log}/product/tablemaintenance/tablemaintenance.log

出力パターン

利用可能なパターン文字列

テーブルメンテナンス操作ログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	×	*ログレベル*
%logger	×	*ロガー名*
%message	○	*ログメッセージ*
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

テーブルメンテナンス操作ログで利用可能なMDCキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
vctm.connect_id	○	データベース接続ID
vctm.tenant_database	○	アクセスしたデータベースの種類
vctm.tablename	○	テーブル名
vctm.statement_type	○	SELECT/INSERT/UPDATE/DELETE
vctm.sql_result	×	INSERT, UPDATE, DELETEの場合:処理件数 SELECTの場合:取得されたレコードの内容 ※出力されるデータ量が膨大になる可能性があります

テーブルメンテナンスインポート・エクスポートログ

テーブルメンテナンス インポート・エクスポート 実行結果出力ログは、テーブルメンテナンスのインポート、またはエクスポートを実行したときに実行結果に対して出力を行います。
初期設定におけるテーブルメンテナンスインポート・エクスポートログの識別名は「TABLE_EDIT.IMPORT_EXPORT」、実行結果出力ログを出すように設定してあります。
実行結果詳細ログを出力する場合は、テーブルメンテナンスインポート・エクスポートログの識別子を「TABLE_EDIT.IMPORT_EXPORT」から「TABLE_EDIT.IMPORT_EXPORT.FILE」に変更してください。
このログが出力される条件は、テーブルメンテナンス画面、およびジョブスケジューラからのテーブル・インポート、およびテーブル・エクスポートを実行した場合に出力されます。

> **注意**
>
> テーブルメンテナンスインポート・エクスポートログは、 intra-mart Accel Platform 2015 Summer(Karen) 以降で利用可能です。

モジュール	TableMaintenance
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_tablemaintenance.xml

```
<included>

 <appender name="TABLE_EDIT.IMPORT_EXPORT" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <file>${im.log}/product/tablemaintenance/import_export.log</file>
  <append>true</append>

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/product/tablemaintenance/import_export%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="ch.qos.logback.classic.PatternLayout">
    <!--
     vctm.connect_id: データベース接続ID
     vctm.statement_type: 行われた操作の種類。下記のいずれかです。
       ・インポート(import)
       ・エクスポート(export)
     vctm.filepath: インポート・エクスポート時に利用もしくは生成されたアーカイブデータのパス
      ※TABLE_EDIT.IMPORT_EXPORT.FILEが有効になっていない場合は出力されません。
     ログメッセージ本体(message)
       ・インポートの場合
        インポート対象のテーブル名と、処理されたレコード数が表示されます
          ・エラーとなった件数(error)
          ・新規挿入された件数(insert)
          ・更新された件数(update)
          ・削除された件数(delete)
        例：　[sample_age error:0, insert:0, update:3, delete:0]
       ・エクスポートの場合
        エクスポート対象のテーブル名が表示されます。
        例：　sample_age, sample_population, sample_prefecture
    -->
    <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] [%X{user.cd}] [%X{vctm.connect_id}] [%X{vctm.statement_type}]
[%X{vctm.filepath}] %message%n</pattern>
   </layout>
  </encoder>
 </appender>

 <!--
  下記の操作に対するログ出力を行います。
   ・レコード編集画面におけるインポートとエクスポート
   ・テーブル・インポート
   ・テーブル・エクスポート
   ・テーブル・インポート（ジョブスケジューラ）
   ・テーブル・エクスポート（ジョブスケジューラ）
  上記の操作ログは、infoレベルで出力されます。
 -->
 <logger name="TABLE_EDIT.IMPORT_EXPORT" additivity="false">
  <level value="info" />
  <appender-ref ref="TABLE_EDIT.IMPORT_EXPORT" />
 </logger>

 <!--
  下記のデータファイルをパブリックストレージに保存します。
   ・インポート時に使用されたデータ
   ・エクスポート時に出力されたデータ

  infoレベルで有効になります。

  データファイルは以下のディレクトリに出力されます。
   ・インポートの場合
    %パブリックストレージ%/products/tablemaintenance/log/%データベース接続ID%/import
   ・エクスポートの場合
    %パブリックストレージ%/products/tablemaintenance/log/%データベース接続ID%/export
 -->
 <logger name="TABLE_EDIT.IMPORT_EXPORT.FILE" additivity="false">
  <level value="off" />
  <appender-ref ref="TABLE_EDIT.IMPORT_EXPORT" />
 </logger>
</included>
```

標準出力設定

ログレベル（初期値）	INFO
出力先（初期値）	ファイル - `${im.log}/product/tablemaintenance/import_export.log`

出力パターン

利用可能なパターン文字列

テーブルメンテナンスインポート・エクスポートログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	×	*ログレベル*
%logger	×	*ロガー名*
%message	○	*ログメッセージ* importの場合、インポート対象のテーブル名と、処理されたレコード数が表示されます。 exportの場合、エクスポート対象のテーブル名が表示されます。
%X	○	*MDC* 利用可能なキーは「*利用可能なMDCキー*」を参照してください。

利用可能なMDCキー

テーブルメンテナンスインポート・エクスポートログで利用可能なMDCキーは以下の通りです。
MDCについては「*MDC*」を参照してください。

MDCキー	設定有無（初期値）	説明
vctm.connect_id	○	データベース接続ID
vctm.statement_type	○	実際に行われた操作(import/export)

vctm.filepath	○	ログの識別名が「TABLE_EDIT.IMPORT_EXPORT」の場合、表示されません。 ログの識別名が「TABLE_EDIT.IMPORT_EXPORT.FILE」の場合、以下の情報が出力されます。 ＊import:インポート元となるファイル名 ＊export:エクスポート先となるファイル名 このMDCキーに紐づいて出力されるファイルのパスはパブリックストレージからの相対パスに対応しています。

IM-LogicDesignerログ

IM-LogicDesignerログにはIM-LogicDesignerのロジックフローの実行中の情報が出力されます。

ロジックフローの実行時のフェーズと、そのフェーズの中で扱っている変数についての詳細は、ログレベル DEBUG でログが出力されます。

> **注意**
> IM-LogicDesignerログは、intra-mart Accel Platform 2015 Winter(Lydia) 以降で利用可能です。

モジュール	IM-LogicDesigner
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_logic_flow.xml

```
<included>

  <!--
    - IM-Logic LOGIC_FLOW_LOG
   -->
  <appender name="LOGIC_FLOW_LOG_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${im.log}/platform/logic_flow.log</file>
    <append>true</append>

    <!--
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>
        ${im.log}/platform/logic_flow-%d{yyyy-MM-dd}.log
      </fileNamePattern>
    </rollingPolicy>
    -->

    <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
      <fileNamePattern>${im.log}/platform/logic_flow%i.log</fileNamePattern>
      <minIndex>1</minIndex>
      <maxIndex>5</maxIndex>
    </rollingPolicy>

    <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
      <maxFileSize>10MB</maxFileSize>
    </triggeringPolicy>

    <encoder>
      <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %-5level %logger{255} - %msg%n</pattern>
    </encoder>
  </appender>

  <!-- Formerly Known As Logger Name "LOGIC_FLOW_LOG" -->
  <logger name="LOGIC_FLOW_LOG" additivity="false">
    <level value="off" />
    <appender-ref ref="STDOUT" />
    <appender-ref ref="LOGIC_FLOW_LOG_FILE" />
  </logger>

</included>
```

標準出力設定

ログレベル（初期値）	OFF
出力先（初期値）	ファイル - ${im.log}/platform/logic_flow.log

出力パターン

利用可能なパターン文字列

IM-LogicDesignerログで利用可能なパターン文字列は以下の通りです。
パターン文字列については「*パターン文字列*」を参照してください。

フォーマット文字列	設定有無（初期値）	説明
%d	○	*出力日時*
%thread	○	*スレッド名*
%level	○	*ログレベル*
%logger	○	*ロガー名*
%msg	○	*ログメッセージ*

スマートメニューランキングログ

スマートメニューランキングログは、スマートメニューランキングデータを生成するために遷移情報を記録することを目的としたログです。

注意
スマートメニューランキングログは、intra-mart Accel Platform 2017 Winter(Rebecca) 以降で利用可能です。

モジュール	スマートメニューランキング
設定場所	%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_smart_menu_ranking.xml

```
<included>

 <!--
   - SMART_MENU_RANKING_LOG
  -->
 <appender name="SMART_MENU_RANKING_FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
  <filter class="jp.co.intra_mart.system.smart_menu.ranking.log.filter.SmartMenuRankingLogFilter" />
  <file>${im.log}/platform/smart_menu_ranking.log</file>
  <append>true</append>

  <!--
  <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
   <fileNamePattern>
    ${im.log}/platform/smart_menu_ranking-%d{yyyy-MM-dd}.log
   </fileNamePattern>
  </rollingPolicy>
  -->

  <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
   <fileNamePattern>${im.log}/platform/smart_menu_ranking%i.log</fileNamePattern>
   <minIndex>1</minIndex>
   <maxIndex>5</maxIndex>
  </rollingPolicy>

  <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
   <maxFileSize>10MB</maxFileSize>
  </triggeringPolicy>

  <encoder class="ch.qos.logback.core.encoder.LayoutWrappingEncoder">
   <layout class="jp.co.intra_mart.system.smart_menu.ranking.log.layout.SmartMenuRankingPatternLayout" />
  </encoder>
 </appender>

   <!-- DON'T CHANGE -->
 <logger name="TRANSITION_LOG" additivity="false">
  <appender-ref ref="SMART_MENU_RANKING_FILE" />
 </logger>
   <!-- DON'T CHANGE -->

</included>
```

標準出力設定

ログレベル	*画面遷移ログ* の設定に依存

出力先（初期値）	ファイル - ${im.log}/platform/im_smart_menu_ranking.log

> **注意**
> スマートメニューランキングログのログファイルの出力先やローテートの方法を変更する場合は合わせてスマートメニューランキング設定のログ設定を変更する必要があります。
> 詳細は、設定ファイルリファレンスの「スマートメニューランキング設定」を参照してください。

出力パターン

スマートメニューランキングログはレイアウトクラスである jp.co.intra_mart.system.smart_menu.ranking.log.layout.SmartMenuRankingPatternLayout にて決められた形式で出力されます。

> **注意**
> ログの出力パターンの変更を行った場合、スマートメニューランキングのランキングデータの集計が正常に動作しません。

バーチャルテナントでのログ運用

バーチャルテナント環境におけるログの運用について説明します。

ログにテナントIDを出力する

バーチャルテナント機能を利用し、バーチャルテナントによる複数テナントを運用する場合
ログにテナントIDを加えて出力することで、どのテナントの処理内容かを判断できます。
intra-mart Accel Platform では、MDCにテナントIDの情報を保持しています。
詳細は「 *システムログ* 」「 *特定用途ログ* 」の *利用可能なMDCキー* 内の tenant.id を参照ください。

テナント単位にログを分割する

標準のログ設定では、システムログ・特定用途ログには各テナントの情報が混在して出力されます。
テナントを複数運用する場合、情報が混在しログの解析や問題の切り分けが困難になる場合があります。
その場合、ログファイルをテナント単位で分割することを検討してください。

ログファイルの分割は、「 *MDCに格納されている値でログを分割する* 」で紹介している方法を用い、MDCのテナントIDをキーとして設定することで可能です。
以下にセキュリティログの設定例を示します。

```
<included>

  <!-- appender "SECURITY_FILE" の設定を省略しています。 -->

  <appender name="SECURITY_BY_TENANT" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
      <key>tenant.id</key>
      <defaultValue>_NO_TENANT_</defaultValue>
    </discriminator>
    <timeout>30 minutes</timeout>
    <sift>
      <appender name="SECURITY_FILE-${tenant.id}" class="ch.qos.logback.core.rolling.RollingFileAppender">
        <file>${im.log}/platform/${tenant.id}/security.log</file>
        <append>true</append>

        <rollingPolicy class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
          <fileNamePattern>${im.log}/platform/${tenant.id}/security%i.log</fileNamePattern>
          <minIndex>1</minIndex>
          <maxIndex>5</maxIndex>
        </rollingPolicy>

        <triggeringPolicy class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
          <maxFileSize>10MB</maxFileSize>
        </triggeringPolicy>

        <encoder>
          <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %X{log.report.sequence} %-5level %logger{255} %X{tenant.id} %X{log.id} -
%X{security.id.session} %X{security.id.account} %X{security.id.usertype} [%X{log.message.code}] %msg %X{request.id}%nopex%n</pattern>
        </encoder>
      </appender>
    </sift>
  </appender>

  <logger name="SECURITY_LOG" additivity="false">
    <level value="warn" />
    <appender-ref ref="STDOUT" />
    <appender-ref ref="SECURITY_FILE" />
    <appender-ref ref="SECURITY_BY_TENANT" />
  </logger>

</included>
```

> **注意**
>
> テナント毎にログを分割して出力する場合、出力するログの数×テナント数でファイルのオープンが行われます。
> その為、設定によっては多くのログファイルがオープンされ、ファイルディスクリプタが不足してしまう可能性があります。
> これを回避するためには、OSレベルで利用可能なファイルディスクリプタ数の上限を変更するといった対応を検討してください。

項目

付録

パターン文字列

パターン文字列はログ設定ファイルでパターンの指定を行う箇所で利用できます。
`%変換文字列` のように指定します。
例）%logger

ロガー名

変換文字列	説明
c{length} **lo{length}** **logger{length}**	ロガー名を表示します。 length を指定することで、出力するロガー名の長さを制限します。 例） mainPackage.sub.sample.Bar というロガー名を出力する場合

変換文字列	出力結果
%logger	mainPackage.sub.sample.Bar
%logger{10}	m.s.s.Bar
%logger{15}	m.s.sample.Bar

出力日時

変換文字列	説明
d{pattern} **date{pattern}** **d{pattern, timezone}** **date{pattern, timezone}**	ログの出力を行った日時を表示します。 pattern を指定することで出力する日時の表示形式を変更することが可能です。 pattern で指定可能な形式は java.text.SimpleDateFormat と同様です。 pattern を指定しない場合は yyyy-MM-dd HH:mm:ss,SSS で表示します。 timezone を指定することで、特定のタイムゾーンでの日時で表示することが可能です。 timezone で指定可能な形式は、Asia/Tokyo などの java.util.Timezone#getTimeZone(String) で指定する値と同様です。 timezone を指定しない場合は、Java VMのデフォルトのタイムゾーンで表示します。 例） 日本時間 2013年10月1日 1時2分3秒4ミリ秒にログを出力した場合 ※ Java VMのデフォルトのタイムゾーンは Asia/Tokyo とします。

変換文字列	出力結果
%d	2013-10-01 01:02:03,004
%d{yyyy/MM/dd}	2013/10/01
%d{yyyy/MM/dd, GMT}	2013/09/31
%d{'yyyy-MM-dd HH:mm:ss,SSS', GMT}	2013-09-31 16:02:03,004

> **コラム**
>
> timezone の指定を行いつつ、pattern に , （カンマ）を含む形式を指定する場合は、pattern の文字列を ' （クォート）で括ってください。

ログメッセージ

変換文字列	説明
m **msg** **message**	ログメッセージを表示します。

ログレベル

変換文字列	説明
p **le** **level**	ログレベルを表示します。

改行

変換文字列	説明
n	改行します。

経過時間

変換文字列	説明
r **relative**	ログを生成してからの経過時間を表示します。単位はミリ秒です。

スレッド名

変換文字列	説明
t **thread**	ログを生成したスレッド名を表示します。

MDC

変換文字列	説明
X{key} **MDC{key}** **X{key:-default}** **MDC{key:-default}**	ログが生成したスレッドのMDCの内容を表示します。 key にMDCのキーを指定します。 default を指定した場合、指定したキーにの値が null であった場合に default の値が出力されます。 MDCの詳細については「*MDC*」を参照してください。

例外

変換文字列	説明
ex{depth} **exception{depth}** **throwable{depth}** **ex{depth, evaluator...}** **exception{depth, evaluator...}** **throwable{depth, evaluator...}**	例外を出力します。 depth に値を指定した場合の動作は以下の通りです。 **値** / **出力内容** **short** — 最初のスタックトレースのみ表示します。ネストしたスタックトレースを表示しません。 **full** — ネストしたスタックトレースをすべて表示します。depth を指定しない場合のデフォルト値です。 **整数** — スタックトレースを指定した深さまで表示します。 evaluator の指定方法についてはLogbackのWebサイト「パターン文字列ex」を参照してください。

パッケージ情報付き例外

変換文字列	説明
xEx{depth} **xException{depth}** **xThrowable{depth}** **xEx{depth, evaluator...}** **xException{depth, evaluator...}** **xThrowable{depth, evaluator...}**	例外を出力します。 「*例外*」とは異なり、スタックトレースにクラスが含まれているjarファイルが表示されます。 depth に値を指定した場合の動作は以下の通りです。 **値** / **出力内容** **short** — 最初のスタックトレースのみ表示します。ネストしたスタックトレースを表示しません。 **full** — ネストしたスタックトレースをすべて表示します。depth を指定しない場合のデフォルト値です。 **整数** — スタックトレースを指定した深さまで表示します。 evaluator の指定方法についてはLogbackのWebサイト「パターン文字列xEx」を参照してください。

例外の抑止

変換文字列	説明
nopex **nopexception**	例外を無視し、例外の表示を抑制します。

置換

変換文字列	説明
replace(p){r, t}	対象の文字列の置換を行います。 `p` で指定した文字列に含まれる `r` をすべて `t` に変換します。 `r` は正規表現で記述します。 例） ロガー名の `.`（ドット）を `/`（スラッシュ）に置換する **ロガー名** jp.co.intra_mart.common.platform.log.Logger **`%logger{10}` による出力** j.c.i.c.p.l.Logger **`%replace(%logger{10}){'\.`, `\/'}` による出力** j/c/i/c/p/l.Logger

逆表示例外

変換文字列	説明
rEx{depth} **rootException{depth}** **rEx{depth, evaluator...}** **rootException{depth, evaluator...}**	例外を発生元を先頭で出力します。 「*例外*」や「*パッケージ情報付き例外*」ではネストした例外の「外側」から「内側」の順に表示しますが、このパターン文字列では「内側」から「外側」の順に表示します。 `depth` に値を指定した場合の動作は以下の通りです。 **値** **出力内容** **short** 最初のスタックトレース（根本の例外）のみ表示します。 **full** ネストしたスタックトレースをすべて表示します。`depth` を指定しない場合のデフォルト値です。 **整数** スタックトレースを指定した深さまで表示します。 `evaluator` の指定方法についてはLogbackのWebサイト「パターン文字列rEx」を参照してください。

ログを解析する

ログ解析方法の例を紹介します。
サーバの運用状態を知りたい場合にシステムログ、リクエストログなどのログを 1 つのファイルにまとめることにより、様々な情報を得ることができます。

> **コラム**
> 解析に必要な情報は、あらかじめログ設定ファイルに設定しておく必要があります。各ログファイルで出力可能な項目については、「*特定用途ログ*」を参照してください。

トランザクション単位での調査

スレッド、出力時間、ログ出力順序番号でソートを行うことにより、ログメッセージを処理トランザクション毎にまとまることができます。

利用者の操作状況の調査

リクエストログ、*セキュリティログ*、*画面遷移ログ*、*マスタデータ更新ログ* をリクエスト ID、出力時間でソートを行うことにより、利用者の操作ログとして扱うことができます。

セットアップ実行ログについて

テナント環境セットアップ、および、サンプルデータセットアップの実行結果は、セットアップ実行結果ログに出力されます。
詳細は、「テナント環境セットアップ 仕様書のセットアップ実行結果ログ」を参照してください。

インポートエラーの原因をログから追跡する

インポートエラーが発生した場合にその原因を追跡する方法を紹介します。

> **注意**
>
> 以下は、「*インポート・エクスポートログ*」の設定が初期設定である前提の説明です。

1. インポートの実行IDを特定します。
   実行IDを特定する方法は以下の通りです。

   1. 実行IDはインポートエラー発生時に標準出力に出力されるエラーログから特定する
      例）アカウントインポート実行エラー時の出力

      ```
      [ERROR] I.j.c.i.f.d.u.ImportExportLog - [E.IWP.IMPORTEXPORT.IMPORTER.10001] データのインポートに失敗しました。 実行クラス =
      jp.co.intra_mart.system.admin.account.data.AccountInfoXmlImporter, 実行ID = <%実行ID%>
      ```

   2. テナント環境セットアップ、または、サンプルデータセットアップにてインポートエラーが発生した場合は、セットアップ実行ログに出力されている実行IDを参照する
      例）アカウントインポート実行エラー時の出力

      ```
      <import-result-detail-data success="false">
        <module-id>im_sample_data</module-id>
        <execute-id><%実行ID%></execute-id>
        <import-type>DML</import-type>
        <importer-id>jp.co.intra_mart.import.StandardAccountXmlImporter</importer-id>
        <target-name>products/import/basic/im_sample_data/im_sample_data-account.xml</target-name>
        <message>[E.IWP.IMPORTEXPORT.IMPORTER.10001] データのインポートに失敗しました。 実行クラス =
      jp.co.intra_mart.system.admin.account.data.AccountInfoXmlImporter, 実行ID = <%実行ID%></message>
      </import-result-detail-data>
      ```

2. インポートログを特定します。
   システムストレージに出力されている、インポートログを特定します。
   ログは `<STORAGE_PATH>/system/storage/log/import-export` フォルダ直下に出力されます。
   ファイル名は `<%実行ID%>.log` です。

3. インポートログからエラー原因を特定します。
   インポートログの付与情報、例外を元にエラー原因を特定します。
   以下は、アカウントインポートジョブの実行により、インポートエラーが発生した際のエラーログです。
   出力されている処理番号（MDCキー：data.io.record.num）から、５番目に処理されたデータがエラーであること、また、出力されている例外情報から、アカウントに指定されているロケールIDの指定が不正であることが読み取れます。

   ```
   [2013-10-18 10:46:19.999] 5i7um1o07vj9k : [INFO ] 0 - [I.IWP.IMPORTEXPORT.IMPORTER.10001] インポートを開始します。 実行クラス =
   jp.co.intra_mart.system.admin.account.data.AccountInfoXmlImporter, 実行ID = a0fd4cdd-0b2d-4ed4-a2db-b86d1971140b : -
   [2013-10-18 10:46:20.179] 5i7um1o080j9t : [ERROR] 5 - [E.IWP.IMPORTEXPORT.IMPORTER.10001] データのインポートに失敗しました。 実行クラス =
   jp.co.intra_mart.system.admin.account.data.AccountInfoXmlImporter, 実行ID = a0fd4cdd-0b2d-4ed4-a2db-b86d1971140b : -
   jp.co.intra_mart.foundation.data.exception.DataImporterException: jp.co.intra_mart.system.data.importer.validation.exception.ValidateException:
   [E.IWP.IMPORTER.ACCOUNT.00001] ロケールマスタに定義されていないロケールIDが指定されています。
    at jp.co.intra_mart.system.admin.account.data.AbstractAccountInfoImporter.validate(AbstractAccountInfoImporter.java:328)
    at jp.co.intra_mart.system.admin.account.data.AbstractAccountInfoImporter.execute(AbstractAccountInfoImporter.java:85)
    at jp.co.intra_mart.system.admin.account.data.AbstractAccountInfoImporter.execute(AbstractAccountInfoImporter.java:44)
    at jp.co.intra_mart.system.data.importer.impl.AbstractDataImporter.importData(AbstractDataImporter.java:153)
    at jp.co.intra_mart.foundation.data.importer.DataImportExecutor.importData(DataImportExecutor.java:189)
    at jp.co.intra_mart.system.job_scheduler.import_export.StandardImportJob.executeImport(StandardImportJob.java:109)
    at jp.co.intra_mart.system.job_scheduler.import_export.StandardImportJob.execute(StandardImportJob.java:91)
    at jp.co.intra_mart.system.job_scheduler.quartz.jobs.AbstractJobnetExecuter.executeJob(AbstractJobnetExecuter.java:288)
    at jp.co.intra_mart.system.job_scheduler.quartz.jobs.SerializeJobnetExecuter.executeJobnet(SerializeJobnetExecuter.java:62)
    at jp.co.intra_mart.system.job_scheduler.quartz.jobs.AbstractJobnetExecuter.execute(AbstractJobnetExecuter.java:70)
    at org.quartz.core.JobRunShell.run(JobRunShell.java:213)
    at jp.co.intra_mart.system.job_scheduler.quartz.threadpool.ExecutorThreadPool$WorkerThread.run(ExecutorThreadPool.java:530)
   Caused by: jp.co.intra_mart.system.data.importer.validation.exception.ValidateException: [E.IWP.IMPORTER.ACCOUNT.00001] ロケールマスタに定義されていないロケールIDが指定されています。
    at jp.co.intra_mart.system.admin.account.data.validate.AccountValidator.localeId(AccountValidator.java:89)
    at jp.co.intra_mart.system.admin.account.data.validate.AccountValidator.validate(AccountValidator.java:59)
    at jp.co.intra_mart.system.admin.account.data.AbstractAccountInfoImporter.validate(AbstractAccountInfoImporter.java:325)
    ... 11 common frames omitted
   ```

JMXによるログ設定の操作

JMX（Java Management Extensions）利用してログレベルの取得、変更を行う方法を紹介します。

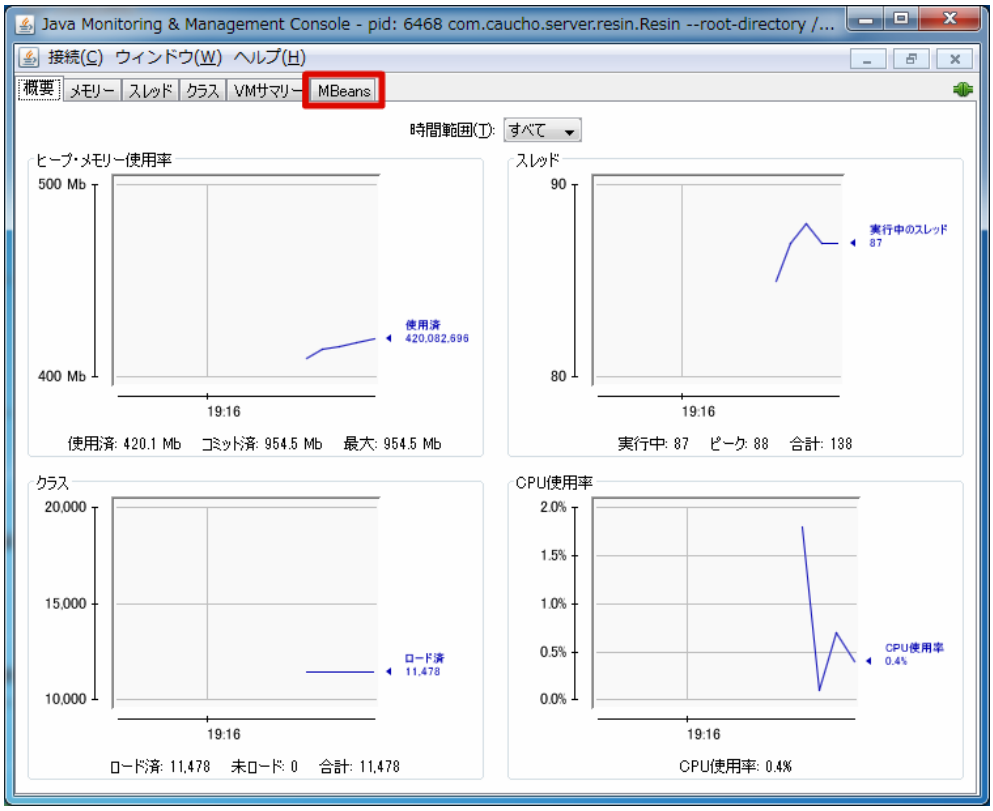
> **コラム**
>
> JMXによる操作を行うには `im_logger.xml` に `<jmxConfigurator>` が設定されている必要があります。
> intra-mart Accel Platform の初期状態では設定されています。

コラム

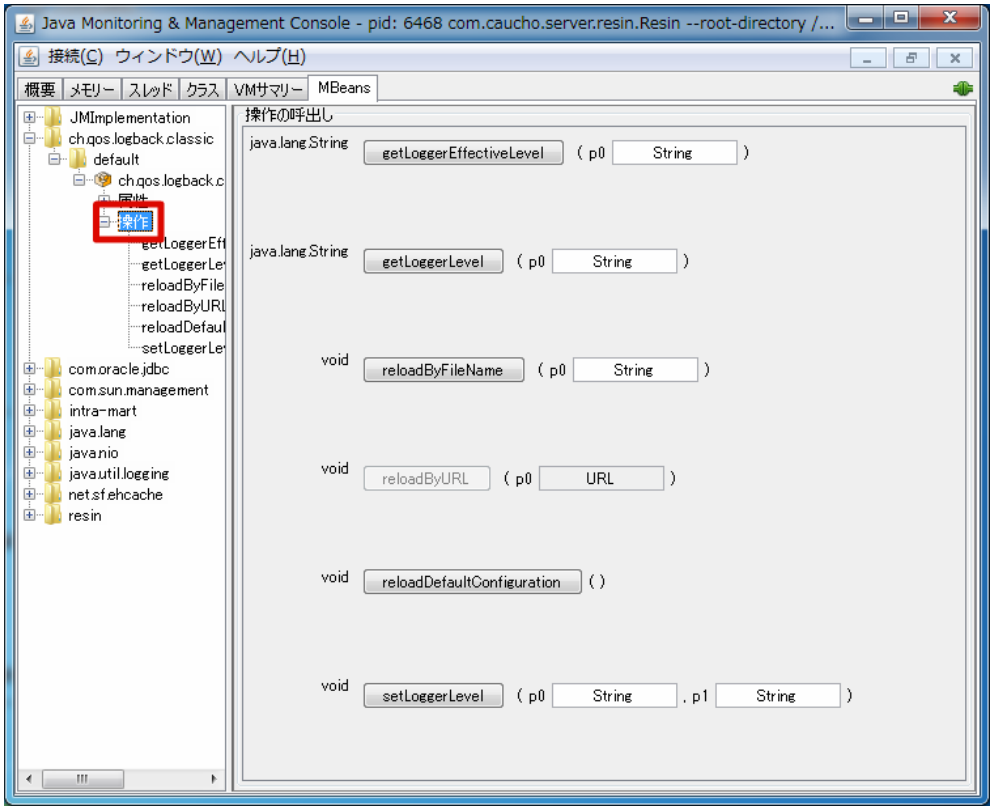
以下は、ローカルプロセスとして起動しているアプリケーションサーバのログ設定を操作する手順です。
外部のリモートプロセスのアプリケーションサーバのログ設定を操作を行う必要がある場合は、適切なJMXを別途行ってください。

1. アプリケーションを起動します。
2. アプリケーションサーバが利用しているJava VMのjconsoleを起動します。
   jconsoleは %JAVA_HOME%/bin 直下に配置されています。
3. 該当するアプリケーションサーバのプロセスを選択し、「接続」をクリックします。

4. 「MBean」タブをクリックします。



5. ch.qos.logback.classic > default > ch.qos.logback.classic.jmx.JMXConfigurator > 操作 をクリックします。



6. 以下の操作が行えます。

   - getLoggerEffectiveLevel

     指定したロガー名を利用した場合に有効となるログレベルが取得できます。
     第１引数にはロガー名を指定します。
     ロガー名を持つロガーが存在しない場合は、ルートロガーのログレベルが取得できます。

   - getLoggerLevel

     指定したロガー名を持つロガーに設定されているログレベルが取得できます。
     第１引数にはロガー名を指定します。
     ロガー名を持つロガーが存在しない場合は、nullが返却されます。

   - setLoggerLevel

     指定したロガー名に対してログレベルを設定します。
     第１引数にはロガー名を指定します。
     第２引数にはログレベルを指定します。

MDCに格納されている値でログを分割する

MDCに格納されている値ごとにログファイルを分割する方法を紹介します。
*セキュリティログ*をMDCキー `security.id.usertype` を利用して、一般ユーザ、システム管理者などのユーザ種別ごとに別々に出力するサンプルを作成します。

1. セキュリティログ設定ファイル（`%CONTEXT_PATH%/WEB-INF/conf/log/im_logger_security.xml`）を以下のように変更します。

```
<included>

  <!-- appender "SECURITY_FILE" の設定を省略しています。 -->

  <appender name="PER_USERTYPE" class="ch.qos.logback.classic.sift.SiftingAppender">
    <discriminator>
      <key>security.id.usertype</key>
      <defaultValue>unknown</defaultValue>
    </discriminator>
    <sift>
      <appender name="FILE-${security.id.usertype}" class="ch.qos.logback.core.FileAppender">
        <file>${im.log}/platform/security_per_usertype/${security.id.usertype}.log</file>
        <append>true</append>
        <encoder>
          <pattern>[%d{yyyy-MM-dd HH:mm:ss.SSS}] [%thread] %X{log.report.sequence} %-5level %logger{255} %X{log.id} -
%X{security.id.session} %X{security.id.account} %X{security.id.usertype} [%X{log.message.code}] %msg %X{request.id}%nopex%n</pattern>
        </encoder>
      </appender>
    </sift>
  </appender>

  <logger name="SECURITY_LOG" additivity="false">
    <level value="warn" />
    <appender-ref ref="STDOUT" />
    <appender-ref ref="SECURITY_FILE" />
    <appender-ref ref="PER_USERTYPE" />
  </logger>

</included>
```

2. アプリケーションを起動（再起動）します。
3. 一般ユーザログイン画面（http://<HOST>:<PORT>/<CONTEXT_PATH>/login）にアクセスし、存在しない一般ユーザのユーザコード／パスワードを入力しログインに失敗します。

   コラム
   ログインに失敗することでセキュリティログ（WARNレベル）を出力します。

4. システム管理者ログイン画面（http://<HOST>:<PORT>/<CONTEXT_PATH>/system/login）にアクセスし、存在しないシステム管理者のユーザコード／パスワードを入力しログインに失敗します。

   コラム
   ログインに失敗することでセキュリティログ（WARNレベル）を出力します。

5. %CONTEXT_PATH%/WEB-INF/log/security_per_usertype 直下に、user.log と administrator.log がそれぞれ出力されています。

   コラム
   一般ユーザでのセキュリティログが user.log に、システム管理者でのセキュリティログが administrator.log に出力されます。

独自に作成したログの実装を利用する

アプリケーション開発者が独自のAppenderやEncoderを作成できます。
独自に作成したクラスが含まれるjarファイルを %CONTEXT_PATH%/WEB-INF/lib 直下に配置することで利用できます。

commons-logging, log4jの利用について

commons-loggingライブラリ、および、log4jライブラリを利用したログ出力を行うアプリケーションも、プログラムを変更すること無くログ出力を行うことができます。
この場合各ライブラリは、*SLF4J* を経由して、*Logback* にてログを出力します。
intra-mart Accel Platform で利用しているSLF4Jのブリッジライブラリのバージョン情報は以下の通りです。

iAP の バージョン	バージョン	jarファイル名
2013 Summer(Damask) 以前	1.6.6	jcl-over-slf4j-1.6.6.jar log4j-over-slf4j-1.6.6.jar
2013 Autumn(Eden) から 2014 Winter(Iceberg) 以前	1.7.5	jcl-over-slf4j-1.7.5.jar log4j-over-slf4j-1.7.5.jar
2015 Spring(Juno) 以降	1.7.10	jcl-over-slf4j-1.7.10.jar log4j-over-slf4j-1.7.10.jar

この方法でログを出力する場合、commons-logging、および、log4jのログ設定は、利用できません。それぞれの設定を intra-mart Accel Platform のログ設定ファイルに移行する必要があります。
ログ設定ファイルの作成方法ついては「*ログ設定*」を参照してください。

SQLログを出力する

log4jdbcを利用する場合

SQLログの出力は、パフォーマンスやデータ量など運用時の各種要件を考慮した上で、DBベンダー提供のツールやOSS等の各種ライブラリを利用してください。
ここでは、「log4jdbc」を利用して intra-mart Accel Platform 上で発行しているSQLをログに出力する方法を紹介します。（log4jdbcに関するお問い合わせは log4jdbcのWebサイトで行ってください。）
以下はアプリケーションサーバ Resin での手順です。

1. 必要な資材を準備します。
   - log4jdbcの取得

     https://code.google.com/p/log4jdbc/downloads/list から log4jdbc4-*.jar をダウンロードします。

   - Janinoライブラリの取得

     intra-mart Accel Platform では Job Scheduler Service が一定の間隔でジョブの監視を行うため、定期的にSQLを発行します。
     このSQLがログへ出力されることを抑制するために「*EvaluatorFilter*」と「*JaninoEventEvaluator*」を利用します。
     https://janino-compiler.github.io/janino/ から、janino-*.jar と commons-compiler-*.jar を取得します。

2. 資材を配置します。
   %CONTEXT_PATH%/WEB-INF/lib 直下に取得した log4jdbc4-*.jar、janino-*.jar、commons-compiler-*.jar を配置します。

3. resin-web.xml を変更します。
   %CONTEXT_PATH%/WEB-INF/resin-web.xml のJDBCドライバの完全修飾クラス名、データベース接続先を変更します。
   - PostgreSQL

     変更前

     ```
     <type>org.postgresql.ds.PGConnectionPoolDataSource</type>
     <url>jdbc:postgresql://localhost:5432/dbname</url>
     ```

     変更後

     ```
     <type>net.sf.log4jdbc.DriverSpy</type>
     <url>jdbc:log4jdbc:postgresql://localhost:5432/dbname</url>
     ```

   - Oracle Database

     変更前

     ```
     <type>oracle.jdbc.pool.OracleConnectionPoolDataSource</type>
     <url>jdbc:oracle:thin:@localhost:1521:orcl</url>
     ```

     変更後

     ```
     <type>net.sf.log4jdbc.DriverSpy</type>
     <url>jdbc:log4jdbc:oracle:thin:@localhost:1521:orcl</url>
     ```

   - Microsoft SQL Server

     変更前

     ```
     <type>com.microsoft.sqlserver.jdbc.SQLServerConnectionPoolDataSource</type>
     <url>jdbc:sqlserver://localhost:1433;DatabaseName=databasename</url>
     ```

     変更後

     ```
     <type>net.sf.log4jdbc.DriverSpy</type>
     <url>jdbc:log4jdbc:sqlserver://localhost:1433;DatabaseName=databasename</url>
     ```

   > コラム
   > ホストやデータベース名は開発環境にあわせて設定してください。

4. ログ設定ファイルを作成する
   %CONTEXT_PATH%/WEB-INF/conf/log 直下に im_logger_log4jdbc.xml を作成し、内容を以下のようにします。

```
<included>

   <appender name="JDBC_STDOUT" class="ch.qos.logback.core.ConsoleAppender">
      <filter class="ch.qos.logback.core.filter.EvaluatorFilter">
         <evaluator>
            <expression>return message.contains("IMJOB_QRTZ_SCHEDULER_STATE") || message.contains("IMJOB_QRTZ_TRIGGERS") ||
message.contains("im_async_task_info");</expression>
         </evaluator>
         <onMatch>DENY</onMatch>
         <onMismatch>NEUTRAL</onMismatch>
      </filter>
      <encoder>
         <outputPatternAsHeader>true</outputPatternAsHeader>
         <pattern>[%level] %logger{10} - %msg%n</pattern>
      </encoder>
   </appender>

   <logger name="jdbc.sqlonly" additivity="false">
      <level value="off" />
   </logger>
   <logger name="jdbc.sqltiming" additivity="false">
      <level value="info" />
      <appender-ref ref="JDBC_STDOUT" />
   </logger>
   <logger name="jdbc.audit" additivity="false">
      <level value="off" />
   </logger>
   <logger name="jdbc.resultset" additivity="false">
      <level value="off" />
   </logger>
   <logger name="jdbc.connection" additivity="false">
      <level value="off" />
   </logger>

</included>
```

5. アプリケーションを起動（再起動）します。
6. コンソール（標準出力）を確認します。
   SQLを利用する処理が動作するごとに以下のようなログが出力されます。

```
[INFO] j.sqltiming - select b.menu_group_id from ( SELECT menu_group_id FROM b_m_menu_group_cate_inclusion WHERE
category IN ('im_sitemap_pc') ) a inner join b_m_menu_group_b b on a.menu_group_id = b.menu_group_id
inner join b_m_menu_b c on b.menu_id = c.menu_id ORDER BY c.sort_number,b.menu_group_id
 {executed in 43 msec}
[INFO] j.sqltiming - SELECT * FROM imaz_resource WHERE resource_id =
'd0b2358a8017b27c11bfacc2e7640f85b47d3f89aeacaaebe3d1c24ba92486be'
 {executed in 0 msec}
[INFO] j.sqltiming - select a.resource_group_set_id, a.resource_group_id from imaz_resource_group a inner join imaz_resource_group_ath
b on a.resource_group_set_id = b.resource_group_set_id and a.resource_group_id = b.resource_group_id
where b.resource_id = 'd0b2358a8017b27c11bfacc2e7640f85b47d3f89aeacaaebe3d1c24ba92486be' order
by a.resource_group_set_id, a.resource_group_id
 {executed in 2 msec}
[INFO] j.sqltiming - select a.resource_group_id, a.resource_group_set_id, c.resource_id, c.uri, c.resource_type
from imaz_resource_group a left join imaz_resource_group_ath b on a.resource_group_id = b.resource_group_id
left join imaz_resource c on b.resource_id = c.resource_id where a.resource_group_id =
'd0b2358a8017b27c11bfacc2e7640f85b47d3f89aeacaaebe3d1c24ba92486be'
order by a.resource_group_set_id, a.resource_group_id, c.resource_type, c.resource_id
 {executed in 1 msec}
```

mirageを利用する場合

mirageフレームワークモジュールの機能を使用することで、バインド変数値付きのSQLログをコンソールに出力できます。

1. `%CONTEXT_PATH%/WEB-INF/conf/log` 直下の `im_logger_mirage.xml` を開きます。
2. SqlExecutor のログレベルを「debug」に変更します。

```
<included>
   <logger name="jp.co.intra_mart.mirage" additivity="false">
      <level value="info" />
      <appender-ref ref="STDOUT" />
   </logger>
   <logger name="jp.co.intra_mart.mirage.SqlExecutor" additivity="false">
      <level value="debug" />
      <appender-ref ref="STDOUT" />
   </logger>
</included>
```

3. アプリケーションを起動（再起動）します。

コラム

このログは、mirageフレームワークを利用した機能からSQL発行を行った場合にのみ出力されます。
スクリプト開発モデルのTenantDatabaseオブジェクト、SharedDatabaseオブジェクト からの実行時などが該当します。

コラム

開発環境での利用を想定しているため、インストール時にはSQLログ出力が行われないように設定されています。
必要に応じてログレベルを変更してください。