OpenJDK 17

OpenJDK 17.0.1 リリースノート

リリースノート

Last Updated: 2021-11-29

OpenJDK 17 OpenJDK 17.0.1 リリースノート

リリースノート

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

本書では、OpenJDK 17 の新機能の概要と、考えられる既知の問題と、その回避策を説明します。

目次

前書き

OpenJDK (Open Java Development Kit) は、Java Platform Standard Edition (Java SE) のオープンソース実装です。

Red Hat ビルドの OpenJDK 向けパッケージは、Red Hat Enterprise Linux および Microsoft Windows で利用でき、Red Hat Ecosystem Catalog の JDK および JRE として同梱されています。

多様性を受け入れるオープンソースの強化

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

RED HAT ドキュメントへのフィードバック

弊社のドキュメントに関するご意見やご感想をお寄せください。フィードバックをお寄せいただくには、ドキュメントのテキストを強調表示し、コメントを追加できます。

本セクションでは、フィードバックの送信方法を説明します。

前提条件

- Red Hat カスタマーポータルにログインしている。
- Red Hat カスタマーポータルで、**マルチページ HTML** 形式でドキュメントを表示します。

手順

フィードバックを提供するには、以下の手順を実施します。

1. ドキュメントの右上隅にある **フィードバック** ボタンをクリックして、既存のフィードバックを確認します。

注記

フィードバック機能は、**マルチページ HTML** 形式でのみ有効です。

2. フィードバックを提供するドキュメントのセクションを強調表示します。
3. ハイライトされたテキスト近くに表示される **Add Feedback** ポップアップをクリックします。ページの右側のフィードバックセクションにテキストボックスが表示されます。
4. テキストボックスにフィードバックを入力し、**Submit** をクリックします。ドキュメントに関する問題が作成されます。
5. 問題を表示するには、フィードバックビューで問題トラッカーリンクをクリックします。

第1章 OPENJDK のサポートポリシー

Red Hat は、その製品で OpenJDK の一部のメジャーバージョンに対応しています。一貫性のため、このバージョンは、長期サポート (LTS) として指定されている Oracle JDK バージョンと引き続き似ています。

Red Hat は、Red Hat が OpenJDK を初めて導入してから 6 年間、OpenJDK のメジャーバージョンに対応しています。

OpenJDK 17 は、2027 年 11 月まで、Microsoft Windows および Red Hat Enterprise Linux で対応しています。

注記

RHEL 6 のライフサイクルは 2020 年 11 月に終了します。OpenJDK 17 は RHEL 6 ではサポートされていません。

関連情報

詳細は、「OpenJDK のライフサイクルおよびサポートポリシー」を参照してください。

第2章 アップストリームの OPENJDK 17 との相違点

Red Hat Enterprise Linux の OpenJDK には、OpenJDK のアップストリームディストリビューションの構造上の変更が数多く含まれています。Windows バージョンの OpenJDK は、Red Hat Enterprise Linux の更新にできる限り従います。

以下は、Red Hat OpenJDK 17 における最も注目すべき変更の一覧です。

- FIPS のサポート。Red Hat OpenJDK 17 は、RHEL システムが FIPS モードであるかどうかを自動的に検出し、そのモードで動作するように OpenJDK 17 を自動的に設定します。この変更は、Microsoft Windows 向けの OpenJDK ビルドには適用されません。

- 暗号化ポリシーのサポート。Red Hat OpenJDK 17 は、Red Hat Enterprise Linux システム設定から、有効な暗号アルゴリズムと鍵サイズの制約の一覧を取得します。これは、TLS、証明書パスの検証、および署名済み JAR で使用されます。さまざまなセキュリティープロファイルを設定して、安全性と互換性のバランスをとることができます。この変更は、Microsoft Windows 向けの OpenJDK ビルドには適用されません。

- Red Hat Enterprise Linux は、アーカイブ形式のサポート用の **zlib**、イメージのサポート用の **libjpeg-turbo**、**libpng**、**giflib** などのネイティブライブラリーと動的にリンクします。また、RHEL はフォントのレンダリングと管理のために、**Harfzbuzz** および **Freetype** に対して動的にリンクします。

- **src.zip** ファイルには、OpenJDK に同梱されるすべての JAR ライブラリーのソースが含まれます。

- Red Hat Enterprise Linux は、タイムゾーン情報のソースとして、システム全体のタイムゾーンデータファイルを使用します。

- Red Hat Enterprise Linux は、システム全体の CA 証明書を使用します。

- Microsoft Windows には、Red Hat Enterprise Linux で利用可能な最新のタイムゾーンデータが含まれています。

- Microsoft Windows は、Red Hat Enterprise Linux から入手可能な最新の CA 証明書を使用します。

関連情報

- システムが FIPS モードであるかどうかの検出の詳細は、Red Hat RHEL Planning Jira Web ページの システム FIPS 検出の改善 の例を参照してください。

- 暗号化ポリシーの詳細は、『Red Hat Enterprise Linux **セキュリティーの強化**ガイド』の「システム全体の暗号化ポリシーの使用」を参照してください。

第3章 非推奨で、サポートされていない OPENJDK の機能

OpenJDK のアップストリームプロジェクトでは、コミュニティーの関心が低く、代替ソリューションが改善されたため、一部のテクノロジーへのサポートが削除されました。

OpenJDK 17 をインストールする前に、以下の非推奨およびサポートされていない機能を確認してください。

アプレットの API

OpenJDK 17 では、一般的な Web ブラウザーでは Applet API の Java プラグインに対応していないため、Applet API が非推奨になりました。OpenJDK 17 では、Applet API に関連するクラスおよび API 要素も非推奨になりました。

非推奨のアプレットAPI の詳細は、「JEP 398 - Applet API for Removal の非推奨 」を参照してください。

同時マークアンドスイープ (CMS) ガーベジコレクター (GC)

OpenJDK 17 では、CMS GC のサポートが削除されました。CMS は、以前のリリースの OpenJDK で非推奨になりました。

CMS ガルーンの削除の詳細は、「JEP 363 - CMS (Concurrent Mark Sweep) ガベージコレクターの削除」を参照してください。

バイアスされたロックの非推奨と無効化

OpenJDK 17 では、ロックとそのコマンドラインオプションについて、バイアスがかかっていました。

非推奨のバイアスロック機能の詳細は、JEP 374: バイアスロックの非推奨および無効化 を参照してください。

Graal JIT コンパイラーおよび Java AOT コンパイラー

OpenJDK 17 では、Graal just-in-time (JIT) コンパイラーが削除されたため、以下のモジュールが OpenJDK 17 から削除されました。

- **jaotc**
- **jdk.aot**
- **jdk.internal.vm.compiler**
- **jdk.internal.vm.compiler.management**

GrallVM は、AOT (ahead-of-time) コンパイルタスクに引き続き使用できます。Mandrel は、GraalVM、OpenJDK、および RHEL の機能を統合するために使用できます。

また、OpenJDK 17 は、**jaotc** ツールとしても知られている Java a ahead-of-time (AOT) コンパイラーを削除します。

外部に構築されたバージョンのコンパイラーに関連する JIT コンパイラータスクには、試験的な Java レベルの JVMCI コンパイラーインターフェース (JVMCI) を引き続き使用できます。

AOT コンパイラーおよび JIT コンパイラーの削除に関する詳細は、JEP 410: Experimental AOT および JIT コンパイラーを削除 を参照してください。

PACK200 ツールおよび API

OpenJDK 17 では、PACK200 ツールおよび PACK200 API のサポートが削除されました。両方の API は、以前の OpenJDK リリースで非推奨となりました。

削除された PACK200 ツールは以下のとおりです。

- **pack200**
- **unpack200**

削除された PACK200 API は以下のとおりです。

- **java.util.jar.Pack200**
- **java.util.jar.Pack200.Packer**
- **java.util.jar.Pack200.Unpacker**

このようなツールや API は、技術的なダウンロード速度が向上し、Pack200 プラグインに対するブラウザーのサポートが減ったため、削除されました。

OpenJDK 互換のアプリケーションの場合は、**jlink** ツールまたは **jパッケージ** ツールを使用して、アプリケーションの JAR を縮小できます。

PACK200 ツールおよび API の削除の詳細は、「JEP 367: Pack200 ツールおよび API の削除 」を参照してください。

jlink ツールの詳細は、『OpenJDK 11 **jlink** を使用した Java ランタイム環境のカスタマイズ ガイド』を参照してください。

Parallel Scavenge アルゴリズムおよび Serial Old GC アルゴリズム

OpenJDK 17 では、Parallel Scavenge アルゴリズムと Serial Old Garbage Collector (GC) アルゴリズムの組み合わせが非推奨になりました。

この GC アルゴリズムの組み合わせの詳細は、「JEP 366: Deprecate the ParallelScavenge + SerialOld GC Combination」を参照してください。

Nashorn JavaScript エンジン

OpenJDK 17 では、Nashorn JavaScript Engine、その API、および **jjs** ツールへの対応が解除されました。

Nashorn JavaScript Engine の削除の詳細は、JEP 372: Remove the Nashorn JavaScript Engine を参照してください。

Remote Method Invocation (RMI)

OpenJDK 17 では、(RMI) のアクティベーションメカニズムおよびその **java.rmi.activation** API パッケージが削除されました。OpenJDK 17 では、その他の RMI 機能を引き続き使用できます。

RMI アクティベーションメカニズムの削除の詳細は、「JEP 407: Remove RMI Activation 」を参照してください。

セキュリティー管理者

OpenJDK 17 では、コミュニティーの関心が低かったため、クラスとそのメソッドを含む Security Manager 機能が非推奨になりました。

セキュリティーマネージャーの非推奨に関する詳細は、「JEP 411: Deprecate the Security Manager for Removal」を参照してください。

Solaris ポートおよび SPARC ポート

OpenJDK 17 では、ソースコードが削除され、Solaris/SPARC ポート、Solaris/x64 ポート、および Linux/SPARC ポートがサポートされるようになりました。

Solaris ポートおよび SPARC ポートの削除の詳細は、「JEP 381: Remove the Solaris and SPARC Ports」を参照してください。

値ベースのクラス警告

OpenJDK 17 では、プリミティブラッパークラスを値ベースとして指定し、そのコンストラクターは非推奨になりました。これにより、Java プラットフォームで値ベースのクラスの同期が間違って試行されることを示す新しい非推奨の警告が作成されます。

値ベースのクラスの警告の詳細は、JEP 390: Warnings for Value-Based Classes を参照してください。

第4章 OPENJDK の新機能

OpenJDK 17 には、Java アプリケーションの使用を強化する新機能が同梱されています。

OpenJDK 17 には、以下の新機能が含まれます。

コンテキスト固有のデシリアライズフィルター

OpenJDK 17 は、JVM 全体のフィルターファクトリーを使用して、コンテキスト固有で動的に選択されたデシリアライズフィルターを設定する Java プログラム機能を提供します。Java プログラムは、このフィルターをファクトリーで起動して、各デシリアライズ操作のフィルターを選択します。

コンテキスト固有のデシリアライズフィルターの詳細は、JEP 415: Context-Specific Deserialization Filters を参照してください。

Edwards-Curve デジタル署名アルゴリズム (EdDSA)

OpenJDK 17 は、EdDSA アルゴリズムを使用した暗号化署名の実装に対応しています。

EdDSA の詳細は、「JEP 339: Edwards-Curve Digital Signature Algorithm (EdDSA) 」を参照してください。

ファイルマッピングモード

OpenJDK 17 には、**FileChannel** API への JDK 固有のファイルマッピングモードが含まれているため、NVM (Non-volatile Memory) にマッピングする**MappedByteBuffer** インスタンスを作成できます。

JDK 固有のファイルマッピングモードの詳細は、JEP 352: Non-Volatile Mapped Byte Buffers を参照してください。

外部リンカー API (Incubator 機能)

OpenJDK 17 には、Foreign Linker API が導入されました。この API により、以下の機能を備えた Java プログラムが提供されます。

- 静的に型付けされた Java からネイティブコードへのアクセス。
- ネイティブライブラリーへのバインドが単純化されました。

Foreign Linker API の詳細は、JEP 389: Foreign Linker API を参照してください。

外部メモリーアクセス API (Incubator 機能)

OpenJDK 17 には、Java プログラムが、Java ヒープの外にある外部メモリーにアクセスするのに安全かつ効率的に使用できる Foreign-Memory Access API が導入されました。

Foreign-Memory Access API の詳細は、JEP 393: Foreign-Memory Access API を参照してください。

Foregin Function とメモリー API (Incubator 機能)

OpenJDK 17 には、Foreign Function と Memory API が導入されました。Java プログラムは、この API を使用して、Java ランタイム外のコードおよびデータと対話できます。

JVM の外部のコードなどの外部関数を効率的に呼び出し、JVM が管理しないメモリーなどの外部メモリーに安全にアクセスすることで、API は以下の機能を提供します。

- Java プログラムがネイティブライブラリーを呼び出せるようにします。
- 一般的な JNI (Java Native Interface) の問題が発生することなく、ネイティブデータを処理します。

Foregin Function およびメモリーAPI の詳細は、JEP 412: Foreign Function & Memory API を参照してください。

非表示のクラス

OpenJDK 17 は、非表示のクラスに対応しています。他のクラスのバイトコードでは、このような非表示のクラスを使用できません。

フレームワークは、ランタイム時にこのようなクラスを生成し、**反射** と呼ばれるプロセスを介して間接的に使用できます。非表示のクラスは、アクセス制御ネストのメンバーとして定義できます。

非表示のクラスの詳細は、JEP 371: Hidden Classes を参照してください。

HotSpot クラスメタデータメモリー

OpenJDK 17 は、メタスペースなどの未使用の HotSpot クラスメタデータメモリーを、オペレーティングシステムに速やかに返します。これにより、オフヒープのメモリー使用率が低くなり、メタスペースコードの量が単純になります。

Hotspot クラスメタデータメモリーの詳細は、JEP 387: Elastic Metaspace を参照してください。

Java パッケージ

OpenJDK 17 には、新しいパッケージ**java.lang.invoke.constant**が同梱されています。このパッケージには、クラスファイルの名義の説明や、定数プールから読み込むことができる定数などのランタイムアーティファクトをモデル化するのに使用できる API が含まれます。

java.lang.invoke.constant の詳細は、「JEP 334: JVM Constants API」を参照してください。

JDK Flight Recorder (JFR) データ

OpenJDK 17 は、アプリケーションのパフォーマンスを継続的に監視するのに使用できるデータストリーム形式で、JFR データを提供します。

JFR データストリームの詳細は、JEP 349: JFR Event Streaming を参照してください。

jpackage ツール

OpenJDK 17 には、自己完結型の Java アプリケーションのパッケージ化に使用できる**jpackage** ツールが同梱されています。

jpackage ツールの詳細は、JEP 392: Packaging Tool を参照してください。

instanceof 演算子のパターンマッチング

instanceof 演算子は、パターンマッチングに対応します。パターンマッチングは、プログラム内の共通論理に対応します。**instanceof** 演算子は、パターンマッチングを使用して、簡潔で安全な方法でオブジェクトからコンポーネントを条件付きで抽出できます。

パターンマッチングの詳細は、JEP 394: Pattern Matching for **instanceof** を参照してください。

擬似乱数生ジェネレーター

OpenJDK 17 には、追加のインターフェースタイプと、疑似乱数ジェネレーター (PRNG) の実装が含まれます。Java プログラムは、このようなインターフェースタイプのいずれかを使用して、既存の PRNG での重複コードを減らしたり、その他のアプリケーションで PRNG アルゴリズムを再利用したりできます。このようなインターフェースの種類は、以下のとおりです。

- **SplittableRandomGenerator**
- **JumpableRandomGenerator**
- **LeapableRandomGenerator**

- **ArbitrarilyJumpableRandomGenerator**

新しいインターフェースタイプの詳細は、「JEP 356: Enhanced Pseudo-Random Number Generators」を参照してください。

レコードのクラス

レコードクラスを使用すると、Java コードを拡張できます。レコードクラスは、不変データの透過的なクラスとして機能するため、レコードクラスは、クラスを宣言するためのコンパクトな構文を提供します。

レコードクラスの詳細は、JEP 395: Records を参照してください。

シールされたクラス

シールドされたクラスとそのインターフェースを使用して、Java コードを拡張できます。シールされたクラスとそのインターフェースは、他のクラスやインターフェースがそれらを拡張または実装できるものを制限します。

シールされたクラスの詳細は、JEP 409: Sealed Classes を参照してください。

テキストブロック

OpenJDK 17 には、テキストブロックが含まれます。テキストブロックは、以下の機能を提供する複数行の文字列リテラルです。

- 文字列のフォーマットを自動的かつ予測可能に設定します。
- ユーザーに、文字列のフォーマットを指定するオプションを提供します。
- Java コードでエスケープシーケンスが必要なくなります。

テキストブロックの詳細は、「JEP 378: Text Blocks」を参照してください。

ベクトル API (Incubator 機能)

OpenJDK 17 では、インキュベーターモジュール **jdk.incubator.vector** を最初に繰り返して、ランタイムに確実にコンパイルするベクトル計算を表現します。

ベクトル計算は、対応している CPU アーキテクチャーでベクトルハードウェア命令を最適化するため、スカラー計算を使用するアーキテクチャーと比較すると、このアーキテクチャーのパフォーマンスが向上します。

ベクトル API の詳細は、「JEP 414: Vector API」を参照してください。

Unix ドメインソケットチャネル

OpenJDK 17 では、**java.nio.channels** パッケージのソケットチャネル API およびサーバーソケットチャネル API に、Unix ドメイン、**AF_UNIX**、ソケットに対応しています。これにより、継承されたチャネルメカニズムが、Unix ドメインのソケットチャネルおよびサーバーソケットチャネルに対応するように拡張されました。

UNIX ドメインソケットチャネルの詳細は、「JEP 380: Unix-Domain Socket Channels」を参照してください。

Z ガーベジコレクター (ZGC)

OpenJDK 17 には、低レイテンシーのコレクターとして使用できる製品機能として ZGC が同梱されています。OpenJDK 17 の ZGC は、スレッドスタックの処理をセーフポイントから同時フェーズに移動します。これは、Shenandoah GC の場合と似ています。

また、OpenJDK 17 では、テクノロジープレビュー機能の ZGC に、以下の拡張機能が追加されました。

- ZGC を Windows オペレーティングシステムに接続します。JEP 365: Windows の ZGC を参照してください。
- ヒープメモリーから、アイドル状態のオペレーティングシステムに、コミットされていない未使用のメモリーを自動的に返します。JEP 351: ZGC: Uncommit Unused Memory を参照してください。

重要

テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat では、実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、「テクノロジプレビュー機能のサポート範囲」を参照してください。

ZGC の詳細は、EP 377: ZGC: A Scalable Low-Latency Garbage Collector を参照してください。

同時スレッドスタック処理の詳細は、JEP 376: ZGC: Concurrent Thread-Stack Processing を参照してください。

第5章 OPENJDK の機能強化

OpenJDK 17 では、以前のリリースの OpenJDK で作成された機能に拡張が行われました。

CDS (Dynamic Class Data Sharing) アーカイブ

OpenJDK 17 は、Java プログラムの実行終了時にクラスの動的アーカイブを可能にするアプリケーションクラスデータ共有を拡張します。アーカイブされたクラスに、読み込み済みのアプリケーションクラスとライブラリークラスがすべて含まれるようになりました。このようなクラスは、デフォルトのベースレイヤーの CDS アーカイブでは利用できませんでした。

ダイナミック CDS アーカイブの詳細は、「JEP 350: Dynamic CDS Archives 」を参照してください。

浮動小数点演算

OpenJDK 17 では、浮動小数点演算が **strictfp** セマンティックと厳密に一致するように設定されています。これは、従来の **strictfp** セマンティックのデフォルト設定と、微妙に異なるデフォルトの floating-point セマンティックからの変更です。

strictfp セマンティックは、Java SE 1.2 で厳格な浮動小数点モードおよびデフォルトの浮動小数点モード以前に導入されたセマンティックに一致する言語および仮想マシンに、元の浮動小数点セマンティックを復元します。

浮動小数点演算の詳細は、JEP 306: Restore Always-Strict Floating-Point Semantics を参照してください。

G1 ガーベジコレクター (GC)

OpenJDK 17 では、以下の機能で G1GC が強化されました。

- これらのコレクションが設定した一時停止時間を超えた場合は、混合コレクションを中止します。
- NUMA (Non-Uniform Memory Access) 対応のメモリー割り当てを実装します。
- ヒープメモリーから、アイドル状態のオペレーティングシステムに、コミットされていない未使用のメモリーを自動的に返します。

混在したコレクションの中止の詳細は、「JEP 344: Abortable Mixed Collections for G1 」を参照してください。

G1GC、JEP 345: NUMA-Aware Memory Allocation for G1 への NUMA 認識メモリー割り当ての詳細は、を参照してください。

G1GC からの未使用のコミット済メモリーの返却の詳細は、「JEP 346: Promptly Return Unused Committed Memory from G1」を参照してください。

レガシー**DatagramSocket API**

OpenJDK 17 では、**java.net.DatagramSocket** API および **java.net.MulticastSocket** API の基本的な実装が、新しい実装に置き換わりました。新しい実装には、以下の機能強化が含まれます。

- 仮想スレッドとの互換性が向上しました。
- メンテナンスとデバッグの提供が簡単になりました。
- 実装が単純になりました。

Legacy DatagramSocket API の詳細は、JEP 373: Reimplement the Legacy DatagramSocket API を参照してください。

仮想スレッドの詳細は、「Loom - Fibers, Continuations and Tail-Calls for the JVM 」を参照してください。

レガシーソケット API

OpenJDK 17 は、**java.net.Socket** API および **java.net.ServerSocket** API で使用される実装を、新しい実装に置き換えます。新しい実装には、以下の機能強化が含まれます。

- ファイバーなどのユーザーモードのスレッドとの互換性が改善されました。
- メンテナンスとデバッグの提供が簡単になりました。
- 実装が単純になりました。

レガシーソケット API の再導入の詳細は、JEP 353: Reimplement the Legacy Socket API を参照してください。

ファイバーの詳細は、「Loom - Fibers, Continuations and Tail-Calls for the JVM 」を参照してください。

Helpful NullPointerExceptions

OpenJDK 17 では、例外の発生時に**null** 値がどこで発生したかを正確に記述することで、JVM (Java 仮想マシン) が生成した**NullPointerException** 例外の使用を改善しています。

NullPointerException例外の改善の詳細は、「JEP 358: Helpful NullPointerExceptions 」を参照してください。

Shenandoah ガベージコレクター

OpenJDK 17 では、Shenandoah ガーベジコレクター (GC) が改善されました。Shenandoah が、Java スレッドと同時に実行することで、GC の一時停止時間を削減できるようになりました。

OpenJDK 17 では、Shenandoah GC でサブミリ秒の一時停止時間を実現できます。Shenandoah での一時停止時間は、Java ヒープサイズから独立しています。

Shenandoah GC の詳細は、JEP 379: Shenandoah: A Low-Pause-Time Garbage Collector を参照してください。

JDK インターナルを強力にカプセル化します。

OpenJDK 17 では、**sun.misc.Unsafe** などの重要な内部 API を除き、デフォルトで JDK の内部要素に対して強力なカプセル化メカニズムを提供しています。

以前のリリースの OpenJDK では、critical 要素や、non-critical 要素など、すべての内部 JDK 要素のカプセル化のレベルを下げることができました。OpenJDK 17 の内部要素では、このレベルのカプセル化を減らすことができません。

JDK の内蔵エレメントを強くカプセル化する方法は、JEP 403: Strongly Encapsulate JDK Internals を参照してください。

switch 表現

OpenJDK 17 は、**switch** ステートメントの機能を拡張し、ステートメント形式または式形式で使用できるようになりました。この機能により、コーディングが簡素化され、**switch** 式を使用したパターンマッチング機能が利用できるようになります。

どちらの形式も、**traditional**、**simplified** のスコープ作成、および制御フロービヘイビアーに対応します。また、両方の形式で、従来の **case... :** ラベルまたは新しい **case...** → ラベルを使用できます。この場合、新しいステートメントでは、**switch** 式の値を指定します。

また、OpenJDK 17 は、**switch** へのパターンマッチングの拡張に対応しています。これにより、複数のパターンに対して、それぞれ特定の動作で式をテストして、複雑なデータ指向のクエリーを簡潔かつ安全に実行できるようになります。OpenJDK 17 の場合、この機能はテクノロジープレビューとしてのみ利用できます。

重要

OpenJDK 17 では、パターンマッチングを **switch** に拡張することがテクノロジープレビュー機能のみになります。テクノロジープレビュー機能は Red Hat の実稼働環境でのサービスレベルアグリーメント (SLA) ではサポートされていないため、Red Hat では実稼働環境での使用を推奨していません。Red Hat では、実稼働環境での使用を推奨していません。これらの機能は、近々発表予定の製品機能をリリースに先駆けてご提供することにより、お客様は機能性をテストし、開発プロセス中にフィードバックをお寄せいただくことができます。Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、「テクノロジプレビュー機能のサポート範囲」を参照してください。

switch 表現機能の詳細は、「JEP 361: Switch Expressions」を参照してください。

Java 関連のテクノロジープレビュー機能の詳細は、JEP 12: Preview Features を参照してください。

第6章 本リリースに関連するアドバイザリー

以下のアドバイザリーは、本リリースに含まれるバグ修正および CVE の修正に発行されています。

- RHSA-2021:4135-01
- RHEA-2021:4136-04
- RHSA-2021:4532-01
- RHSA-2021:4531-01

改訂日時: 2021-11-14 16:29:08 +1000