Red Hat build of Keycloak 24.0

アップグレードガイド

Last Updated: 2024-06-21

法律上の通知

概要

このガイドは、Red Hat build of Keycloak を 22.0.x から 24.0.5 にアップグレードするためのガイドです。

目次

多様性を受け入れるオープンソースの強化

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

第1章 RED HAT BUILD OF KEYCLOAK のアップグレード

このガイドでは、Red Hat build of Keycloak をバージョン 22.0.x からバージョン 24.0.5 にアップグレードする方法について説明します。このガイドの手順は、次の順序で使用してください。

1. 前バージョンの Red Hat build of Keycloak からの移行の変更を確認します。
2. Red Hat build of Keycloak サーバーをアップグレードします。
3. Red Hat build of Keycloak アダプターをアップグレードします。
4. Red Hat build of Keycloak 管理クライアントをアップグレードします。

Red Hat Single Sign-On 7.6 のお客様の場合は、まず 移行ガイド を使用して Red Hat build of Keycloak 22.0 にアップグレードしてください。その後、このアップグレードガイドを使用してください。

第2章 移行の変更

2.1. ユーザープロファイルの変更

2.1.1. ユーザープロファイルがデフォルトで有効

ユーザープロファイル機能がデフォルトで有効になりました。ユーザープロファイルが有効であると想定されるため、**declarative-user-profile** 機能が使用できなくなりました。したがって、**User Profile Enabled** スイッチが **Realm Settings** から削除され、**Unmanaged attributes** に置き換えられました。以前のバージョンから移行する場合、動作は次のようになります。

- **User Profile Enabled** が **ON** に設定されているデプロイメントの場合、アップグレード後に **Unmanaged attributes** が **OFF** に設定されます。その結果、ユーザープロファイルによって明示的にサポートされているユーザー属性のみが許可されます。

- **User Profile Enabled** が **OFF** (**declarative-user-profile** 機能が無効なデプロイメントのデフォルト設定。この機能はデフォルトで無効でした) に設定されているデプロイメントの場合、アップグレード後に **Unmanaged attributes** が **ON** に設定されます。そのため、動作は、ユーザープロファイルが無効になっている以前のバージョンと基本的に同じです。**Attributes** タブは、管理コンソールのユーザー詳細部分に引き続き存在します。また、特定のカスタムテーマがサポートしている限り、ユーザーが登録ページやプロファイル更新ページなどの UI ページを通じて任意の属性を設定できるようになりました。カスタムテーマも以前と同じように動作します。ただし、ユーザープロファイルを使用するようにテーマを更新し、カスタム属性を追加する必要があったカスタムテーマを削除することを検討してください。テーマについては後続のセクションを参照してください。また、**Unmanaged attributes** を **OFF** に切り替えるか、このスイッチを管理者に対してのみ有効にして、管理対象の属性を主に使用することを検討してください。

Unmanaged attributes の詳細は、ユーザープロファイルのドキュメント を参照してください。

2.1.2. デフォルトの検証

デフォルトのユーザープロファイル設定には、基本的な定義済みフィールドに対するデフォルトの一連の検証が付属しています。これらの検証は、**declarative-user-profile** 機能がデフォルトで無効になっていた以前のバージョンには存在しませんでした。下位互換性による問題がある場合は、必要に応じてデフォルトのバリデーターを変更できます。デフォルトのバリデーターは次のとおりです。

- **username**、**email**、**firstName**、**lastName** 属性の最大長は 255 文字です。これらの検証は、最大長が 255 文字のフィールドの **USER_ENTITY** テーブルに対するデータベース制約のため、以前のバージョンでも間接的に存在していました。ただし、ユーザーのストレージプロバイダーを使用する場合は、より長い値を使用できる可能性があります。

- **username** 属性の最小長は 3 文字です。また、ユーザー名には、**username-prohibited-characters** と **up-username-not-idn-homograph** のバリデーターがデフォルトで備わっています。これは以前のバージョンには存在しなかったものです。これらの属性の詳細は、ユーザープロファイルドキュメントの検証セクション を参照してください。レルムスイッチの **Edit username** が有効になっていない限り、ユーザー名はデフォルトでは編集できないことに注意してください。この変更により、既存のユーザーはユーザー名が正しくなくても引き続き機能し、ユーザー名の更新が強制されなくなります。一方、新しいユーザーは、管理 REST API によって登録時または作成時に正しいユーザー名を使用するように強制されます。

- **firstName** 属性と **lastName** 属性には、**person-name-prohibited-characters** バリデーターがあります。これは以前のバージョンには存在しなかったものです。これらの属性の詳細は、ユーザープロファイルドキュメントの検証セクション を参照してください。デフォルトで

は名と姓の両方が編集可能であることに注意してください。そのため、以前のバージョンで正しくない名/姓を使用していたユーザーは、ユーザープロファイルを更新するときにそれらを更新する必要があります。

2.1.3. 不自然な文字を含むユーザー属性名

以前のバージョンでは、**some:attribute** や **some/attribute** などの属性名を使用してユーザーを作成できました。ユーザープロファイル設定では、このような不自然な名前の属性を作成することを意図的に禁止しています。そのため、レルムに対して **Unmanaged attributes** を設定し、(理想的には) 管理者または (本当に必要な場合) エンドユーザーに対して管理対象外の属性を有効にする必要がある場合があります。ただし、このような属性名の使用は回避することを強く推奨します。

2.1.4. 必須アクションのプロファイル検証がデフォルトで有効

新しいレルムでは、必須アクション **verify-profile** がデフォルトで有効になっています。ただし、以前のバージョンから移行した場合、既存レルムの **verify-profile** アクションの状態は、以前と同じになります。以前のバージョンではデフォルトで無効であったため、通常は無効になります。この必須アクションの詳細は、ユーザープロファイルのドキュメント を参照してください。

2.1.5. ユーザープロファイルとレルムに基づいてページをレンダリングする Freemarker テンプレートの変更

このリリースでは、次のテンプレートが更新され、レルムに設定されたユーザープロファイル設定に基づいて属性を動的にレンダリングできるようになりました。

- **login-update-profile.ftl**
- **register.ftl**
- **update-email.ftl**

これらのテンプレートは、それぞれ、更新プロファイル (ユーザーに対して必須アクション **Update Profile** が有効になっている場合)、登録、メール更新 (**UPDATE_EMAIL** 機能が有効になっている場合) ページのレンダリングを担当します。

カスタムテーマを使用してこれらのテンプレートを変更すれば、コンテンツのみが更新されるため、テンプレートが期待どおりに動作します。ただし、宣言型ユーザープロファイル を設定する方法を確認し、可能であればこの機能によって提供されるすべての機能を使用して組み込みテンプレートの変更を回避することを推奨します。

また、このリリースでは、同じフローのページをレンダリングするために **declarative-user-profile** 機能によって使用されるテンプレートが不要になったため、削除されました。

- **update-user-profile.ftl**
- **register-user-profile.ftl**

以前のリリースで、上記のテンプレートをカスタマイズして **declarative-user-profile** 機能を使用していた場合は、それに応じて **login-update-profile.ftl** と **register.ftl** を更新してください。

2.1.6. ブローカー経由で初めてログインしたときのプロファイル更新ページ用の新しい Freemarker テンプレート

このリリースでは、ユーザーが **idp-review-user-profile.ftl** テンプレートを使用して初めてブローカー経由で認証するときに、サーバーが更新プロファイルページをレンダリングします。

以前のリリースでは、最初のブローカーログインフロー中にプロファイルを更新するために使用されたテンプレートは **login-update-profile.ftl** でした。これは、ユーザーがレルムに認証するときにプロファイルを更新するために使用されるものと同じでした。

同じテンプレートを共有するのではなく、各フローに個別のテンプレートを使用すると、どのフローにテンプレートが使用されるかがより明確に区別されます。ただし、特定のフローのページにのみ影響する予期しない変更や動作が発生する可能性があります。

ブローカー経由の認証時にユーザーがプロファイルを更新する方法をカスタマイズするために **login-update-profile.ftl** テンプレートをカスタマイズした場合は、変更内容を新しいテンプレートに必ず移してください。

2.2. テーマの変更

2.2.1. ウェルカムテーマの変更

'ウェルカム' テーマが新しいレイアウトを使用するように更新されました。現在は PatternFly 3 ではなく PatternFly 5 を使用しています。テーマを拡張する場合、または独自のテーマを提供する場合は、次のように更新する必要がある場合があります。

2.2.1.1. PatternFly 3 から PatternFly 5 への移行

ウェルカムテーマは、Red Hat build of Keycloak の中で最も古いテーマの 1 つでした。元々は PatternFly 3 をベースにしていましたが、メジャーバージョンを 1 つスキップして PatternFly 5 を使用するように更新されました。カスタムテーマによって組み込みテーマを拡張する場合は、PatternFly 5 構文を使用するようにカスタムテーマを更新する必要があります。詳細は、PatternFly 5 のドキュメント を参照してください。

独自のカスタムテーマで PatternFly 3 をまだ使用している (組み込みテーマを拡張していない) 場合は、引き続き使用できます。ただし、PatternFly 3 のサポートは今後のリリースで削除されるため、できるだけ早く PatternFly 5 への移行を検討してください。

2.2.1.2. 管理コンソールへの自動リダイレクト

管理コンソールが有効になっている場合は、管理ユーザーがすでに存在すると、ウェルカムページが自動的に管理コンソールにリダイレクトされます。この動作は、**theme.properties** ファイルで **redirectToAdmin** を設定することで変更できます。デフォルトでは、このプロパティーは **false** に設定されていますが、組み込みテーマを拡張する場合は、このプロパティーを **true** に設定します。

2.2.1.3. documentUrl および displayCommunityLinks プロパティーの削除

これらのプロパティーは、以前はナビゲーション要素に使用されていましたが、現在は存在しません。組み込みテーマを拡張すると、これらのプロパティーは効果がなくなるため、**theme.properties** ファイルから削除する必要があります。

2.2.1.4. アセットが 'common' リソースからロードされるようになる

背景、ロゴ、ファビコンなどの画像が、テーマリソースではなく、'common' リソースからロードされるようになりました。このように変更されたため、組み込みテーマを拡張してこれらの画像を上書きする場合は、画像をテーマの 'common' リソースに移動し、**theme.properties** ファイルを更新して新しいパスを含める必要があります。

```
# This defaults to 'common/keycloak' if not set.
import=common/your-theme-name
```

2.2.2. Account Console テーマのカスタマイズの変更

以前に、現在非推奨の Account Console テーマのバージョン 2 を拡張していた場合は、新しい Account Console テーマのバージョン 3 を使用するようにテーマを更新する必要があります。Account Console テーマの新しいバージョンでは、カスタマイズ方法に関していくつかの変更が加えられています。最初から始めるには、新しい カスタマイズクイックスタート に従ってください。

カスタムテーマを移動するには、まず **parent** を新しいテーマに変更します。

```
# Before
parent=keycloak.v2

# After
parent=keycloak.v3
```

カスタムの React コンポーネントがある場合は、相対パスを使用するのではなく、React を直接インポートします。

```
// Before
import * as React from "../../../../common/keycloak/web_modules/react.js";

// After
import React from "react";
```

テーマのカスタマイズに **content.json** を使用している場合は、ファイルの構造にいくつかの変更点があります。具体的には次の点です。

- **content** プロパティーの名前が **children** に変更になりました。
- **id**、**icon**、および **componentName** プロパティーが削除されました。**modulePath** が同じ機能を提供しているためです。

2.2.3. テーマの言語ファイルのデフォルト設定が UTF-8 エンコード

このリリースでは、リソースバンドルファイルを UTF-8 でエンコードすることを想定している Java 以降の標準メカニズムに準拠するようになりました。

以前のバージョンの Keycloak では、**# encoding: UTF-8** のようなコメントを使用して最初の行にエンコーディングを指定することがサポートされていました。これはサポートされなくなり、無視されます。

テーマのメッセージプロパティーファイルが UTF-8 エンコードで読み取られるようになり、自動的に ISO-8859-1 エンコードにフォールバックされます。別のエンコードを使用している場合は、ファイルを UTF-8 に変換してください。

2.3. OPERATOR の変更

2.3.1. Operator が参照するリソースのポーリング

Keycloak CR 経由で参照される Secret と ConfigMap は、API サーバーによって監視されるのではなく、変更をポーリングされるようになりました。このポーリングでは、変更が検出されるまでに 1 分かかる場合があります。

この変更は、これらのリソースに対するラベル操作の必要性を回避するために行われました。アップグレード後、Secret にまだ operator.keycloak.org/component ラベルが付いている場合は、削除または無視される可能性があります。

2.3.2. Operator のカスタマイズプロパティーキー

Operator が詳細設定に使用するプロパティーキーが、**operator.keycloak** から **kc.operator.keycloak** に変更になりました。

2.3.3. Operator の -secrets-store Secret

以前のバージョンの Operator は、監視対象の Secret を追跡するために Secret を作成していました。新しいバージョンの Operator は、-secrets-store Secret を使用しなくなったため、この Secret は削除できます。

2.4. KEYCLOAK CR の変更

2.4.1. Keycloak CR の resources オプション

Keycloak CR および KeycloakRealmImport CR で **resources** オプションが指定されていないと、デフォルト値が使用されます。Keycloak デプロイメントおよびレルムインポートジョブのデフォルトの **requests** メモリーは **1700MiB** に設定され、**limits** メモリーは **2GiB** に設定されます。

2.4.2. デフォルトの Keycloak CR ホスト名

OpenShift 上で実行する場合は、Ingress が有効で、spec.ingress.classname が openshift-default に設定されている場合、Keycloak CR の spec.hostname.hostname を未設定のままにしておくことができます。Operator は、明示的なホストのない OpenShift ルートによって作成されるホスト名と同様に、保存されたバージョンの CR にデフォルトのホスト名 ingress-namespace.appsDomain を割り当てます。appsDomain を変更した場合、または何らかの理由で別のホスト名が必要な場合は、Keycloak CR を更新してください。

2.5. 機能の変更

--features リストと **--features-disabled** リストの両方に同じ機能を指定することができなくなりました。機能は 1 つのリストにのみ指定できます。

--features リストで **docker** などのバージョンなしの機能名を使用すると、サポート期間が最も長い最新の機能バージョンを有効にできます。リリースに関する動作をより予測可能なものにする必要がある場合は、代わりに **docker:v1** などの特定のバージョンを参照してください。

2.6. その他の変更点

2.6.1. Admin API コンテキストと Account コンテキストの両方におけるユーザー表現の変更

org.keycloak.representations.idm.UserRepresentation と **org.keycloak.representations.account.UserRepresentation** の両方の表現クラスが変更されました。これにより、ルートユーザー属性 (**username**、**email**、**firstName**、**lastName**、**locale** など) が、それぞれ Admin API と Account API から表現ペイロードを取得または送信するときに一貫した表現を使用するようになりました。

username、**email**、**firstName**、**lastName**、**locale** 属性が、新しい **org.keycloak.representations.idm.AbstractUserRepresentation** ベースクラスに移動しました。

なお、**getAttributes** メソッドはカスタム属性のみを表すことを目的としたものです。そのため、このメソッドによって返されるマップにルート属性が含まれることを期待しないでください。このメソッドは主に、特定のユーザーのカスタム属性を更新または取得するクライアントを対象としています。

ルート属性を含むすべての属性を解決するために、新しい **getRawAttributes** メソッドが追加されました。これにより、結果のマップにルート属性も含まれるようになりました。ただし、このメソッドは表現ペイロードからは利用できません。ユーザープロファイルを管理するときにサーバーによって使用されることを目的としたものです。

2.6.2. ビルド時のオプションの **https-client-auth**

オプション **https-client-auth** は、実行時のオプションとして扱われていましたが、これは Quarkus ではサポートされていません。代わりに、このオプションはビルド時に処理する必要があります。

2.6.3. オフラインセッションとリモートセッションの順次ロード

このリリースから、Red Hat build of Keycloak クラスターの最初のメンバーが、リモートセッションを並行してではなく順番にロードするようになります。オフラインセッションのプリロードが有効になっている場合は、それらも順番にロードされます。

以前のコードでは、起動時にクラスター全体でリソース消費量が多くなり、実稼働環境での分析が困難で、ロード中にノードが再起動した場合に複雑な障害が発生する可能性がありました。そのため、セッションの順次ロードに変更になりました。

このバージョンおよび以前のバージョンの Red Hat build of Keycloak では、デフォルトでオフラインセッションをオンデマンドでロードします。これにより、オフラインセッションが多数ある場合は、それらを並行してプリロードするよりもスケーラビリティーが向上します。このデフォルト設定を使用するセットアップは、オフラインセッションのロードストラテジーの変更による影響を受けません。オフラインセッションの事前ロードが有効なセットアップは、オフラインセッションのプリロードが無効なセットアップに移行する必要があります。

2.6.4. Infinispan メトリクスがキャッシュマネージャーとキャッシュ名のラベルを使用

Red Hat build of Keycloak の組み込みキャッシュのメトリクスを有効にしたときに、メトリクスがキャッシュマネージャーとキャッシュ名のラベルを使用するようになりました。

ラベルのない古いメトリクスの例

```
vendor_cache_manager_keycloak_cache_sessions_statistics_approximate_entries_in_memory{cache=
sessions",node="..."}
```

ラベル付きの新しいメトリクスの例

```
vendor_statistics_approximate_entries_in_memory{cache="sessions",cache_manager="keycloak",node
"..."}
```

インストールに対する変更を元に戻すには、カスタム Infinispan XML 設定を使用して、次のように設定を変更します。

```
<metrics names-as-tags="false" />
```

2.6.5. ユーザー属性値の長さの拡張

このリリース以降、Red Hat build of Keycloak は、以前は制限されていた 255 文字を超えるユーザー属性値の保存と検索をサポートします。

ユーザーがアカウントコンソールなどを使用して属性を更新できるセットアップでは、検証を追加してサービス拒否攻撃を防止してください。管理対象外の属性が許可されていないこと、およびすべての編集可能な属性に入力長を制限する検証があることを確認してください。

管理対象外の属性の場合、最大長は 2048 文字です。管理対象の属性の場合、デフォルトの最大長は 2048 文字です。管理者は、**length** タイプのバリデーターを追加することでこれを変更できます。

警告

Red Hat build of Keycloak はユーザー関連のオブジェクトを内部キャッシュにキャッシュします。より長い属性を使用すると、キャッシュによって消費されるメモリーが増加します。したがって、長さ属性のサイズを制限することを推奨します。大きなオブジェクトは Red Hat build of Keycloak の外部に保存し、ID または URL で参照することを検討してください。

この変更により、テーブル **USER_ATTRIBUTE** および **FED_USER_ATTRIBUTE** に新しいインデックスが追加されます。これらのテーブルに 300000 を超えるエントリーが含まれている場合、Red Hat build of Keycloak は、デフォルトで自動スキーマ移行中にインデックスの作成をスキップし、Red Hat build of Keycloak の起動後に手動で適用できるように、代わりに移行中にコンソールに SQL ステートメントを記録します。異なる制限を設定する方法の詳細は、アップグレードガイド を参照してください。

2.6.5.1. LDAP の追加移行手順

これは、以下のすべての条件を満たすインストールが対象です。

- LDAP ディレクトリー内のユーザー属性が 2048 文字を超えているか、バイナリー属性が 1500 バイトを超えている。
- 管理者またはユーザーが、管理コンソール、API、またはアカウントコンソールを介して属性を変更する。

UI および REST API を介してこれらの属性を変更できるようにするには、次の手順を実行します。

1. 上記で特定された属性を、レルムのユーザープロファイルで管理対象の属性として宣言します。
2. 前のステップで追加した各属性に対して、属性値の必要な最小長と最大長を指定して **length** バリデーターを定義します。バイナリー値の場合は、Red Hat build of Keycloak 内部のバイナリー値 base64 エンコーディング用のオーバーヘッドとして、予想されるバイナリー長に 33 パーセントを追加します。

2.6.5.2. カスタムのユーザーストレージプロバイダーの追加移行手順

これは、以下のすべての条件を満たすインストールが対象です。

- Red Hat build of Keycloak のデータベースとして MariaDB または MySQL を実行している。
- テーブル **FED_USER_ATTRIBUTE** のエントリーの **VALUE** 列に 2048 文字を超える内容が含まれている。このテーブルは、フェデレーションが有効になっているカスタムユーザープロバイダーに使用されます。
- 管理者またはユーザーが、管理コンソールまたはアカウントコンソールを介して長い属性を変更する。

UI および REST API を介してこれらの属性を変更できるようにするには、次の手順を実行します。

1. 上記で特定された属性を、レルムのユーザープロファイルで管理対象の属性として宣言します。
2. 前のステップで追加した各属性に対して、属性値の必要な最小長と最大長を指定して **length** バリデーターを定義します。

2.6.6. Admin send-verify-email API が同じメール検証テンプレートを使用

```
PUT /admin/realms/{realm}/users/{id}/send-verify-email
```

このリリースでは、この API は **executeActions.ftl** の代わりに **email-verification.ftl** テンプレートを使用します。

アップグレード前

```
Perform the following action(s): Verify Email
```

アップグレード後

```
Confirm validity of e-mail address email@example.org.
```

この API を使用したユーザーのメールアドレスの検証方法を変更するために **executeActions.ftl** テンプレートをカスタマイズした場合は、変更内容を新しいテンプレートに移してください。

デフォルトの有効期間の値 (12 時間) をオーバーライドできるように、**lifespan** という新しいパラメーターが導入されます。

以前の動作を希望する場合は、次のように **execute-actions-email** API を使用してください。

```
PUT /admin/realms/{realm}/users/{id}/execute-actions-email

["VERIFY_EMAIL"]
```

2.6.7. パスワードハッシュ化の変更

このリリースでは、パスワードハッシュ化のデフォルトを、パスワード保存に関する OWASP の推奨事項 に合わせて調整しました。

この変更の一環として、デフォルトのパスワードハッシュ化プロバイダーが **pbkdf2-sha256** から **pbkdf2-sha512** に変更になりました。また、**pbkdf2** ベースのパスワードハッシュ化アルゴリズムのデフォルトハッシュ反復回数が次のように変更になりました。

プロバイダー ID	アルゴリズム	以前の反復回数	新しい反復回数
pbkdf2	**PBKDF2WithHmacSHA1**	20.000	1.300.000
pbkdf2-sha256	**PBKDF2WithHmacSHA256**	27.500	600.000
pbkdf2-sha512	**PBKDF2WithHmacSHA512**	30.000	210.000

レルムが **hashAlgorithm** と **hashIterations** を使用してパスワードポリシーを明示的に設定していない場合、新しい設定は次回のパスワードベースのログイン時、またはユーザーパスワードが作成または更新されたときに有効になります。

2.6.7.1. 新しいパスワードハッシュ化設定のパフォーマンス

Intel i9-8950HK CPU (12) @ 4.800GHz を搭載したマシンでのテストで、1000 個のパスワードをハッシュ化した場合は、次の時間差が得られました (3 回の実行の平均)。なお、**PBKDF2WithHmacSHA1** の平均所要時間は、実行時間が長いため、より少ない数のパスワードで計算されました。

プロバイダー ID	アルゴリズム	以前の所要時間	新しい所要時間	差
pbkdf2	**PBKDF2WithHmacSHA1**	122 ms	3.114 ms	+2.992 ms
pbkdf2-sha256	**PBKDF2WithHmacSHA256**	20 ms	451 ms	+431 ms
pbkdf2-sha512	**PBKDF2WithHmacSHA512**	33 ms	224 ms	+191 ms

pbkdf2 プロバイダーのユーザーは、許容できるパフォーマンスを再び得るために、ハッシュ反復回数を明示的に減らす必要がある場合があります。これは、レルムのパスワードポリシーでハッシュ反復回数を明示的に設定することによって実行できます。

2.6.7.2. 全体的な CPU 使用率の増加とデータベースアクティビティーの一時的な増加が予想されることについて

Red Hat のテストでは、パスワードベースのログインあたりの CPU 使用率が 5 倍に増加しました。この値には、変更されたパスワードハッシュ化と変更されていない TLS 接続処理の両方が含まれます。アクセストークンとクライアントクレデンシャルグラントの更新など、Red Hat build of Keycloak による平均化効果により、全体的な CPU 使用率の増加は 2 - 3 倍程度になるはずです。ただし、これはインストールの固有のワークロードによって異なります。

アップグレード後、パスワードベースのログイン中に、ユーザーのパスワードが、新しいハッシュアルゴリズムとハッシュ反復回数を使用して 1 回限りのアクティビティーとして再ハッシュ化され、データベースで更新されます。これにより、Red Hat build of Keycloak の内部キャッシュからユーザーがクリ

アされるため、データベースレベルでの読み取りアクティビティーも増加します。このようなデータベースアクティビティーの増加は、再ハッシュ化されたユーザーのパスワードが増えるにつれて、次第に減少します。

2.6.7.3. 古い pbkdf2-sha256 パスワードハッシュ化の使用を続ける方法

レルムの古いパスワードハッシュ化を保持するには、レルムパスワードポリシーで **hashAlgorithm** と **hashIterations** を明示的に指定します。

- **Hashing Algorithm: pbkdf2-sha256**
- **Hashing Iterations: 27500**

2.6.8. 移行用の JPA プロバイダー設定オプションの名前変更

マップストアの削除後に、次の設定オプションの名前が変更されました。

- **spi-connections-jpa-legacy-initialize-empty** から **spi-connections-jpa-quarkus-initialize-empty** へ
- **spi-connections-jpa-legacy-migration-export** から **spi-connections-jpa-quarkus-migration-export** へ
- **spi-connections-jpa-legacy-migration-strategy** から **spi-connections-jpa-quarkus-migration-strategy** へ

2.6.9. 一時的なロックアウトログのイベントへの置き換え

ブルートフォースプロテクターによってユーザーが一時的にロックアウトされた場合に、新しいイベント **USER_DISABLED_BY_TEMPORARY_LOCKOUT** が発生するようになりました。新しいイベントは情報を構造化された形式で提供するため、ID **KC-SERVICES0053** のログは削除されました。

これは成功イベントであるため、新しいイベントはデフォルトで **DEBUG** レベルでログに記録されます。すべての成功イベントのログレベルを変更するには、サーバー管理ガイドのイベントリスナーの章で説明されているように、**spi-events-listener-jboss-logging-success-level** 設定を使用します。

カスタムアクションまたはカスタムログエントリーをトリガーするには、サーバー開発者ガイド のイベントリスナー SPI の説明に従って、カスタムイベントリスナーを作成します。

2.6.10. Keycloak JS インポートを更新する必要性

Keycloak JS を Red Hat build of Keycloak から直接ロードする場合は、このセクションを無視しても問題ありません。NPM パッケージから Keycloak JS をロードし、Webpack、Vite などのバンドラーを使用している場合は、コードにいくつかの変更を加える必要がある可能性があります。Keycloak JS パッケージは、package.json ファイルの **exports** フィールド を使用するようになりました。つまり、インポートを変更する必要がある可能性があります。

```
// Before
import Keycloak from 'keycloak-js/dist/keycloak.js';
import AuthZ from 'keycloak-js/dist/keycloak-authz.js';

// After
import Keycloak from 'keycloak-js';
import AuthZ from 'keycloak-js/authz';
```

2.6.11. HS256 から HS512 への内部アルゴリズムの変更

Red Hat build of Keycloak が内部トークン (Red Hat build of Keycloak 自体によって消費される JWT、たとえばリフレッシュトークンやアクショントークン) に署名するために使用するアルゴリズムが、**HS256** からよりセキュアな **HS512** に変更されます。**hmac-generated-hs512** という名前の新しいキープロバイダーがレルムに追加されました。移行されたレルムでは、古い **hmac-generated** プロバイダーと古い **HS256** キーが維持され、アップグレード前に発行されたトークンが引き続き検証されることに注意してください。古いトークンが存在しなくなったら、鍵のローテーションガイドライン に従って、**HS256** プロバイダーを手動で削除できます。

2.6.12. 部コンテナー内で実行する場合の異なる JVM メモリー設定

コンテナー内で実行する場合の JVM オプション **-Xms** および **-Xmx** が、**-XX:InitialRAMPercentage** と **-XX:MaxRAMPercentage** に置き換えられました。Red Hat build of Keycloak は、静的な最大ヒープサイズ設定ではなく、コンテナーメモリーの合計の 70% に最大値を指定します。

ヒープサイズはコンテナーの合計メモリーに基づいて動的に計算されるため、コンテナーの **メモリー制限を必ず設定** してください。

警告

メモリー制限が設定されていない場合は、最大ヒープサイズがコンテナーメモリーの合計の 70% まで増加するため、メモリー消費量が急激に増加します。

詳細は、異なるメモリー設定を指定する を参照してください。

2.6.13. OAuth 2.0/OpenID Connect Authentication Response に iss パラメーターを追加

RFC 9207 OAuth 2.0 Authorization Server Issuer Identification 仕様で、セキュアな認証応答を実現するために、OAuth 2.0/OpenID Connect Authentication Response に **iss** パラメーターが追加されました。

過去のリリースにはこのパラメーターはありませんでしたが、仕様の要求に従って、Red Hat build of Keycloak にはデフォルトでこのパラメーターが追加されました。

ただし、一部の OpenID Connect/OAuth2 アダプター、特に Red Hat build of Keycloak の古いアダプターでは、この新しいパラメーターで問題が発生する可能性があります。

たとえば、クライアントアプリケーションへの認証に成功すると、パラメーターは常にブラウザー URL に表示されます。このような場合は、認証応答への **iss** パラメーターの追加を無効にすると役立つことがあります。これは、「古いアダプターとの互換性」 で説明されているとおり、Red Hat build of Keycloak 管理コンソールで、**OpenID Connect Compatibility Modes** を含むセクション内のクライアントの詳細で特定のクライアントに対して行うことができます。専用の **Exclude Issuer From Authentication Response** スイッチが存在します。これをオンにすると、認証応答に **iss** パラメーターが追加されるのを防ぐことができます。

2.6.14. ワイルドカード文字の処理

JPA では検索時にワイルドカード **%** と **_** を使用できますが、LDAP などの他のプロバイダーでは * のみを使用できます。* は、LDAP では自然なワイルドカード文字であるため、すべての場所で機能しま

すが、JPA では検索文字列の先頭と末尾でのみ機能していました。このリリース以降、ワイルドカード文字 * だけが、検索文字列内のすべての場所で、すべてのプロバイダーで一貫して機能するようになりました。JPA の **%** や _ など、特定のプロバイダー内のすべての特殊文字はエスケープされます。引用符を追加した完全一致検索 (例: **"w*ord"**) の動作は、以前のリリースと同じままです。

2.6.15. レルムおよびクライアントロールマッパーによってマッピングされたクレームの値形式の変更

このリリースより前は、**Multivalued** 設定が無効になっていると、レルム (**User Realm Role**) とクライアント (**User Client Role**) の両方のプロトコルマッパーが、文字列化された JSON 配列をマッピングしていました。

しかし、**Multivalued** 設定は、クレームをリストとしてマッピングするか、同じ値リストからの単一の値のみをマッピングするか (設定が無効な場合) を示すものです。

このリリースでは、ロールマッパーとクライアントマッパーが、単一値とマークされている場合 (**Multivalued** が無効な場合) に、ユーザーの有効なロールからの単一の値にマッピングされるようになりました。

2.6.16. ログイン UI のパスワードフィールドの変更

このバージョンでは、パスワード入力を非表示/表示を切り替えるトグルを導入する予定です。

影響を受けるページ:

- login.ftl
- login-password.ftl
- login-update-password.ftl
- register.ftl
- register-user-profile.ftl

一般的に、すべての **<input type="password" name="password"/>** が div 内にカプセル化されるようになりました。input 要素の後に、パスワード入力の表示を切り替えるボタンが続きます。

以前のコードの例:

```
<input type="password" id="password" name="password" autocomplete="current-password"
style="display:none;"/>
```

新しいコードの例:

```
<div class="${properties.kcInputGroup!}">
    <input type="password" id="password" name="password" autocomplete="current-password"
style="display:none;"/>
    <button class="pf-c-button pf-m-control" type="button" aria-label="${msg('showPassword')}"
            aria-controls="password" data-password-toggle
            data-label-show="${msg('showPassword')}" data-label-hide="${msg('hidePassword')}">
        <i class="fa fa-eye" aria-hidden="true"></i>
    </button>
</div>
```

2.6.17. kc.sh とシェルのメタ文字

kc.sh が、パラメーターおよび環境変数 JAVA_OPTS_APPEND と JAVA_ADD_OPENS に対して追加のシェル eval を使用しなくなりました。そのため、二重のエスケープ/引用符の使用を継続すると、パラメーターが誤解されることになります。以下に例を示します。

```
bin/kc.sh start --db postgres --db-username keycloak --db-url
"\"jdbc:postgresql://localhost:5432/keycloak?ssl=false&connectTimeout=30\"" --db-password
keycloak --hostname localhost
```

このような場合は、代わりに単一のエスケープを使用してください。

```
bin/kc.sh start --db postgres --db-username keycloak --db-url
"jdbc:postgresql://localhost:5432/keycloak?ssl=false&connectTimeout=30" --db-password keycloak --
hostname localhost
```

また、この変更により、すべての引数を単一の引用符で囲んだ値を使用して kc.sh を呼び出すことができなくなります。たとえば、以下は使用できなくなりました。

```
bin/kc.sh "start --help"
```

個別の引数を使用する必要があります。

```
bin/kc.sh start --help
```

同様に、以下は使用できなくなりました。

```
bin/kc.sh build "--db postgres"
```

個別の引数を使用する必要があります。

```
bin/kc.sh build --db postgres
```

Dockerfile 実行コマンドでも個別の引数の使用が必要です。

2.6.18. **GroupProvider** の変更

最上位レベルのグループを検索して参照できるようにする新しいメソッドが追加されました。このインターフェイスを実装する場合は、次のメソッドを実装する必要があります。

```
Stream<GroupModel> getTopLevelGroupsStream(RealmModel realm,
                                           String search,
                                           Boolean exact,
                                           Integer firstResult,
                                           Integer maxResults)
```

2.6.19. **GroupRepresentation** の変更

- 特定のグループに含まれるサブグループの数をクライアントに通知するための新しいフィールド **subGroupCount** が追加されました。
- **subGroups** リストが、階層データを要求するクエリーでのみ入力されるようになりました。

- このフィールドは "下から上" へ入力されるため、グループのすべてのサブグループを取得するために使用することはできません。**GroupProvider** を使用するか、**GET {keycloak server}/realms/{realm}/groups/{group_id}/children** からサブグループを要求してください。

2.6.20. Group Admin API の新しいエンドポイント

ページネーションをサポートする特定のグループのサブグループを取得する方法として、エンドポイント **GET {keycloak server}/realms/{realm}/groups/{group_id}/children** が追加されました。

2.6.21. RESTEasy リアクティブ

RESTEasy Classic が利用不可になり、選択することができなくなりました。RESTEasy Classic および **org.jboss.resteasy.spi.*** に含まれる関連パッケージに依存している SPI およびコードは、移行が必要になります。

2.6.22. 部分エクスポートに manage-realm 権限が必要

エンドポイント **POST {keycloak server}/realms/{realm}/partial-export** と、管理コンソールの対応するアクションの実行に必要な権限が、**view-realm** ではなく **manage-realm** になりました。このエンドポイントはレルム設定を JSON ファイルにエクスポートするものであるため、新しい権限のほうが適切です。パラメーター **exportGroupsAndRoles** と **exportClients** は、同じ権限 (**query-groups** と **view-clients**) を引き続き管理します。これらのパラメーターは、それぞれレルムグループ/ロールとクライアントをエクスポートに追加します。

2.6.23. クライアントの有効なリダイレクト URI を常に厳密な文字列照合により比較

バージョン 1.8.0 で、リダイレクト URI をクライアントの指定された有効なリダイレクトと比較する際に、ホスト名とスキームに小文字が導入されました。ただし、これはすべてのプロトコルで完全に機能したわけではありません。たとえば、ホストが **http** では小文字で、**https** では小文字ではない場合がありました。OAuth 2.0 セキュリティーの現在のベストプラクティス では、厳密な文字列照合を使用して URI を比較するよう推奨しています。そのため、Red Hat build of Keycloak はこの推奨事項に従い、現時点では、ホスト名とスキームについても、大文字と小文字を厳密に区別して有効なリダイレクトを比較しています。

古い動作に依存しているレルムの場合は、クライアントの有効なリダイレクト URI に、サーバーによって認識される URI ごとに個別のエントリーが保持されるようになりました。

クライアントを設定する際の手順と詳細度は増加しますが、新しい動作により、よりセキュアなデプロイメントが可能になります。パターンベースのチェックがセキュリティー問題の原因となることが多いためです。チェックの実装方法だけでなく、設定方法も原因となります。

2.6.24. Groups.getSubGroups の briefRepresentation パラメーターの処理を修正

バージョン 23.0.0 で、Groups リソースに新しいエンドポイント getSubGroups ("children") が導入されましたが、パラメーターの briefRepresentation が、サブグループの完全な表現を取得することを意味していました。この意味が、短い表現を返すように変更になりました。

2.6.25. jboss-logging イベントメッセージの変更

jboss-logging メッセージの値が、引用符 (デフォルトでは文字 **"**) で囲まれ、改行を防ぐためにサニタイズされるようになりました。プロバイダーに、新しい動作をカスタマイズできる 2 つの新しいオプション (**spi-events-listener-jboss-logging-sanitize** と **spi-events-listener-jboss-logging-quotes**) があります。たとえば、サニタイズと引用符の付加の両方を回避するには、次のようにしてサーバーを起動します。

```
./kc.sh start --spi-events-listener-jboss-logging-sanitize=false --spi-events-listener-jboss-logging-quotes=none ...
```

オプションの詳細は、すべてのプロバイダー設定 を参照してください。

2.7. 非推奨および削除された機能

2.7.1. トラストストアの非推奨化

spi-truststore-file-* オプションとトラストストア関連のオプション **https-trust-store-*** が非推奨になりました。そのため、トラストストアマテリアルの新しいデフォルトの場所である **conf/truststores** を使用するか、**truststore-paths** オプションを使用して目的のパスを指定してください。詳細は、送信要求用の信頼済み証明書の設定 を参照してください。

spi-truststore-file-hostname-verification-policy プロパティーの代わりに **tls-hostname-verifier** プロパティーを使用する必要があります。

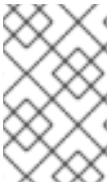
変更の副次的な影響として、トラストストアプロバイダーが常に何らかの証明書で設定されるようになりました (少なくとも、デフォルトの Java 信頼済み証明書が存在します)。この新しい動作は、Red Hat build of Keycloak の他の部分に影響を与える可能性があります。

たとえば、**attestation conveyance** が **Direct** 検証に設定されている場合は、**webauthn** 登録が失敗する可能性があります。以前は、トラストストアプロバイダーが設定されていないと、受信証明書が検証されませんでした。しかし、現在ではこの検証は常に実行されます。この登録は **invalid cert path** エラーで失敗します。ドングルによって送信された証明書チェーンが Red Hat build of Keycloak によって信頼されていないためです。アテステーションを正しく実行するには、オーセンティケーターの認証局がトラストストアプロバイダーに存在している必要があります。

2.7.2. --proxy オプションの非推奨化

--proxy オプションは非推奨となり、今後のリリースで削除される予定です。次の表に、非推奨のオプションとサポートされているオプションの対応関係を示します。

非推奨の使用法	新しい使用法
kc.sh (**proxy** オプションが設定されていない)	**kc.sh**
kc.sh --proxy none	**kc.sh**
kc.sh --proxy edge	**kc.sh --proxy-headers forwarded|xforwarded --http-enabled true**
kc.sh --proxy passthrough	**kc.sh --hostname-port 80|443** (HTTPS が使用されているかどうかによって異なります)
kc.sh --proxy reencrypt	**kc.sh --proxy-headers forwarded|xforwarded**



注記

セキュリティーを強化するため、**--proxy-headers** オプションでは、**forwarded** 値と **xforwarded** 値の両方を同時に選択することはできません (以前の **--proxy edge** および **--proxy reencrypt** の場合と同様)。

警告

プロキシーヘッダーオプションを使用する場合は、リバースプロキシーによって **Forwarded** ヘッダーまたは **X-Forwarded-*** ヘッダーが適切に設定および上書きされることを確認してください。これらのヘッダーを設定するには、リバースプロキシーのドキュメントを参照してください。設定を誤ると、Red Hat build of Keycloak がセキュリティー上の脆弱性にさらされることになります。

Operator を使用するときにプロキシーヘッダーを設定することもできます。

```
apiVersion: k8s.keycloak.org/v2alpha1
kind: Keycloak
metadata:
  name: example-kc
spec:
  ...
  proxy:
    headers: forwarded|xforwarded
```

注記

proxy.headers フィールドが指定されていない場合、Operator はデフォルトで **proxy=passthrough** を暗黙的に設定して、以前の動作にフォールバックします。これにより、サーバーログに非推奨の警告が記録されます。このフォールバックは今後のリリースで削除される予定です。

2.7.3. オフラインセッションのプリロードの非推奨化

Red Hat build of Keycloak のデフォルトの動作では、オンデマンドでオフラインセッションをロードします。起動時にオフラインセッションをプリロードするという従来の動作は、非推奨になりました。起動時にプリロードすると、セッション数の増加に応じて適切にスケールすることができず、Red Hat build of Keycloak のメモリー使用量が増加するためです。古い動作は今後のリリースで削除される予定です。

非推奨でまだ削除されていない古い動作を再度有効にするには、次に示すように機能フラグと SPI オプションを使用します。

```
bin/kc.[sh|bat] start --features-enabled offline-session-preloading --spi-user-sessions-infinispan-preload-offline-sessions-from-database=true
```

UserSessionProvider の API で、メソッド **getOfflineUserSessionByBrokerSessionId (RealmModel レルム、String brokerSessionId)** が非推奨になりました。このメソッドの代わりに、**getOfflineUserSessionByBrokerUserIdStream (RealmModel、String brokerUserId)** を使用し

て、ユーザーのセッションを取得してから、必要に応じてブローカーセッション ID でそれらをフィルタリングします。

2.7.4. データプロバイダーおよびモデルのメソッドの非推奨化

- **RealmModel#getTopLevelGroupsStream()** およびオーバーロードされたメソッドの非推奨化

2.7.5. Cookie の非推奨化と削除

Red Hat build of Keycloak での Cookie 処理のリファクタリングの一環として、Cookie の設定方法にいくつか変更が加えられています。

- 要求がセキュアなコンテキストを介して行われる場合は、すべての Cookie にセキュア属性が設定されるようになりました。
- **WELCOME_STATE_CHECKER** Cookie が **SameSite=Strict** に設定されるようになりました。

カスタムエクステンションの場合は、いくつかの変更が必要になることがあります。

- **LocaleSelectorProvider.KEYCLOAK_LOCALE** は、Cookie が CookieProvider を通じて管理されるようになったため、非推奨になりました。
- **HttpResponse.setWriteCookiesOnTransactionComplete** が削除されました。
- **HttpCookie** が非推奨になりました。代わりに **NewCookie.Builder** を使用してください。
- **ServerCookie** が非推奨になりました。代わりに **NewCookie.Builder** を使用してください。

2.7.6. SAML 暗号化の非推奨モードの削除

バージョン 21 で導入された SAML 暗号化の互換モードが削除されました。システムプロパティー **keycloak.saml.deprecated.encryption** が、サーバーによって管理されなくなりました。暗号化に古い署名鍵をまだ使用しているクライアントは、新しい IDP 設定メタデータから署名鍵を更新する必要があります。

2.7.7. モデルモジュールの名前変更

マップストアの削除後に、次のモジュールの名前が変更になりました。

- **org.keycloak:keycloak-model-legacy-private** を **org.keycloak:keycloak-model-storage-private** に変更
- **org.keycloak:keycloak-model-legacy-services** を **org.keycloak:keycloak-model-storage-services** に変更

org.keycloak:keycloak-model-legacy モジュールが非推奨になりました。次のリリースで **org.keycloak:keycloak-model-storage** モジュールに置き換えられて削除される予定です。

2.7.8. RegistrationProfile フォームアクションの削除

フォームアクション **RegistrationProfile** (認証フローの UI では **Profile Validation** として表示されます) がコードベースから削除され、すべての認証フローからも削除されました。これはデフォルトですべてのレルムの組み込み登録フローに含まれていました。ユーザー属性の検証と、そのユーザーのすべ

ての属性を含むユーザーの作成は、**RegistrationUserCreation** フォームアクションによって処理されるため、**RegistrationProfile** は不要になります。独自のプロバイダーで **RegistrationProfile** クラスを使用していない限り、通常、この変更に関するさらなる対処は不要です。

2.8. 管理ユーザー API を通じてユーザーを更新する際のユーザー属性の部分的な更新がサポート対象外に

管理ユーザー API を介してユーザー属性を更新する場合は、**username**、**email**、**firstName**、**lastName** などのルート属性を含むユーザー属性を更新するときに、部分的な更新ができません。

2.8.1. 非推奨の auto-build CLI オプションの削除

auto-build CLI オプションは、長い間、非推奨とマークされていました。このリリースで完全に削除され、サポートされなくなりました。

start コマンドを実行すると、設定に基づいてサーバーが自動的に構築されます。この動作を防ぐには、**--optimized** フラグを設定します。

2.8.2. イベントの詳細の長さを短縮するオプションの削除

このリリース以降、Keycloak は **EventEntity** 詳細列の長い値をサポートします。そのため、イベントの詳細の長さを短縮するためのオプション **--spi-events-store-jpa-max-detail-length** および **--spi-events-store-jpa-max-field-length** がサポートされなくなりました。

2.8.3. 翻訳からの名前空間の削除

すべての翻訳を admin-ui の 1 つのファイルに移動しました。独自の翻訳を作成したり、admin-ui を拡張したりした場合は、この新しい形式に移行する必要があります。また、データベースに "オーバーライド" がある場合は、キーから名前空間を削除する必要があります。一部のキーは名前空間を除いて同じです。これが最もわかりやすいのは help です。このような場合には、キーの末尾に **Help** が付いています。

必要に応じて、次のノードスクリプトを使用して移行に役立てることができます。このスクリプトは、すべての単一ファイルを取得して新しいファイルに格納し、さらにマッピングの一部を処理します。

```
import { readFileSync, writeFileSync, appendFileSync } from "node:fs";

const ns = [
  "common",
  "common-help",
  "dashboard",
  "clients",
  "clients-help",
  "client-scopes",
  "client-scopes-help",
  "groups",
  "realm",
  "roles",
  "users",
  "users-help",
  "sessions",
  "events",
  "realm-settings",
```

```
  "realm-settings-help",
  "authentication",
  "authentication-help",
  "user-federation",
  "user-federation-help",
  "identity-providers",
  "identity-providers-help",
  "dynamic",
];

const map = new Map();
const dup = [];

ns.forEach((n) => {
  const rawData = readFileSync(n + ".json");
  const translation = JSON.parse(rawData);
  Object.entries(translation).map((e) => {
    const name = e[0];
    const value = e[1];
    if (map.has(name) && map.get(name) !== value) {
      if (n.includes("help")) {
        map.set(name + "Help", value);
      } else {
        map.set(name, value);
        dup.push({
          name: name,
          value: map.get(name),
          dup: { ns: n, value: value },
        });
      }
    } else {
      map.set(name, value);
    }
  });
});

writeFileSync(
  "translation.json",
  JSON.stringify(Object.fromEntries(map.entries()), undefined, 2),
);

const mapping = [
  ["common:clientScope", "clientScopeType"],
  ["identity-providers:createSuccess", "createIdentityProviderSuccess"],
  ["identity-providers:createError", "createIdentityProviderError"],
  ["clients:createError", "createClientError"],
  ["clients:createSuccess", "createClientSuccess"],
  ["user-federation:createSuccess", "createUserProviderSuccess"],
  ["user-federation:createError", "createUserProviderError"],
  ["authentication-help:name", "flowNameHelp"],
  ["authentication-help:description", "flowDescriptionHelp"],
  ["clientScopes:noRoles", "noRoles-clientScope"],
  ["clientScopes:noRolesInstructions", "noRolesInstructions-clientScope"],
  ["users:noRoles", "noRoles-user"],
  ["users:noRolesInstructions", "noRolesInstructions-user"],
  ["clients:noRoles", "noRoles-client"],
```

```
  ["clients:noRolesInstructions", "noRolesInstructions-client"],
  ["groups:noRoles", "noRoles-group"],
  ["groups:noRolesInstructions", "noRolesInstructions-group"],
  ["roles:noRoles", "noRoles-roles"],
  ["roles:noRolesInstructions", "noRolesInstructions-roles"],
  ["realm:realmName:", "realmNameField"],
  ["client-scopes:searchFor", "searchForClientScope"],
  ["roles:searchFor", "searchForRoles"],
  ["authentication:title", "titleAuthentication"],
  ["events:title", "titleEvents"],
  ["roles:title", "titleRoles"],
  ["users:title", "titleUsers"],
  ["sessions:title", "titleSessions"],
  ["client-scopes:deleteConfirm", "deleteConfirmClientScopes"],
  ["users:deleteConfirm", "deleteConfirmUsers"],
  ["groups:deleteConfirm_one", "deleteConfirmGroup_one"],
  ["groups:deleteConfirm_other", "deleteConfirmGroup_other"],
  ["identity-providers:deleteConfirm", "deleteConfirmIdentityProvider"],
  ["realm-settings:deleteConfirm", "deleteConfirmRealmSetting"],
  ["roles:whoWillAppearLinkText", "whoWillAppearLinkTextRoles"],
  ["users:whoWillAppearLinkText", "whoWillAppearLinkTextUsers"],
  ["roles:whoWillAppearPopoverText", "whoWillAppearPopoverTextRoles"],
  ["users:whoWillAppearPopoverText", "whoWillAppearPopoverTextUsers"],
  ["client-scopes:deletedSuccess", "deletedSuccessClientScope"],
  ["identity-providers:deletedSuccess", "deletedSuccessIdentityProvider"],
  ["realm-settings:deleteSuccess", "deletedSuccessRealmSetting"],
  ["client-scopes:deleteError", "deletedErrorClientScope"],
  ["identity-providers:deleteError", "deletedErrorIdentityProvider"],
  ["realm-settings:deleteError", "deletedErrorRealmSetting"],
  ["realm-settings:saveSuccess", "realmSaveSuccess"],
  ["user-federation:saveSuccess", "userProviderSaveSuccess"],
  ["realm-settings:saveError", "realmSaveError"],
  ["user-federation:saveError", "userProviderSaveError"],
  ["realm-settings:validateName", "validateAttributeName"],
  ["identity-providers:disableConfirm", "disableConfirmIdentityProvider"],
  ["realm-settings:disableConfirm", "disableConfirmRealm"],
  ["client-scopes:updateSuccess", "updateSuccessClientScope"],
  ["client-scopes:updateError", "updateErrorClientScope"],
  ["identity-providers:updateSuccess", "updateSuccessIdentityProvider"],
  ["identity-providers:updateError", "updateErrorIdentityProvider"],
  ["user-federation:orderChangeSuccess", "orderChangeSuccessUserFed"],
  ["user-federation:orderChangeError", "orderChangeErrorUserFed"],
  ["authentication-help:alias", "authenticationAliasHelp"],
  ["authentication-help:flowType", "authenticationFlowTypeHelp"],
  ["authentication:createFlow", "authenticationCreateFlowHelp"],
  ["client-scopes-help:rolesScope", "clientScopesRolesScope"],
  ["client-scopes-help:name", "scopeNameHelp"],
  ["client-scopes-help:description", "scopeDescriptionHelp"],
  ["client-scopes-help:type", "scopeTypeHelp"],
  ["clients-help:description", "clientDescriptionHelp"],
  ["clients-help:clientType", "clientsClientTypeHelp"],
  ["clients-help:scopes", "clientsClientScopesHelp"],
  ["common:clientScope", "clientScopeTypes"],
  ["dashboard:realmName", "realmNameTitle"],
  ["common:description", "description"],
];
```

```
mapping.forEach((m) => {
  const key = m[0].split(":");
  try {
    const data = readFileSync(key[0] + ".json");
    const translation = JSON.parse(data);
    const value = translation[key[1]];
    if (value) {
      appendFileSync(
        "translation.json",
        '"' + m[1] + '": ' + JSON.stringify(value) + ',\n',
      );
    }
  } catch (error) {
    console.error("skipping namespace key: " + key);
  }
});
```

これを **public/locale/<language>** フォルダー内の **transform.mjs** というファイルに保存し、次のコマンドで実行します。

```
node ./transform.mjs
```

注記

これでは完全な変換を実行できない可能性がありますが、それに非常に近いものになります。

第3章 RED HAT BUILD OF KEYCLOAK サーバーのアップグレード

アダプターをアップグレードする前にサーバーをアップグレードします。

3.1. アップグレードの準備

サーバーをアップグレードする前に、次の手順を実行します。

手順

1. 設定やテーマなどの古いインストールをバックアップします。
2. 開いているトランザクションを処理し、**data/tx-object-store/** トランザクションディレクトリーを削除します。
3. お使いのリレーショナルデータベースのドキュメントの説明に従い、データベースをバックアップします。
   サーバーをアップグレードすると、データベースは古いサーバーと互換性がなくなります。アップグレードを元に戻す必要がある場合は、まず古いインストールを復元してから、バックアップコピーからデータベースを復元します。

警告

Red Hat build of Keycloak をアップグレードすると、オフラインユーザーセッションを除いて、ユーザーセッションが失われます。ユーザーは再度ログインする必要があります。

3.2. RED HAT BUILD OF KEYCLOAK のダウンロード

アップグレードの準備が整ったら、サーバーをダウンロードできます。

手順

1. Red Hat build of Keycloak の Web サイトから rhbk-24.0.5.zip をダウンロードして展開します。このファイルを展開すると、**rhbk-24.0.5** という名前のディレクトリーが作成されます。
2. このディレクトリーを希望の場所に移動します。
3. **conf/**、**providers/**、**themes/** を以前のインストールから新規インストールにコピーします。

3.3. データベースの移行

Red Hat build of Keycloak では、データベーススキーマを自動的に移行することも、手動で移行することもできます。デフォルトでは、新規インストールを初めて起動すると、データベースが自動的に移行されます。

3.3.1. リレーショナルデータベースの自動移行

自動移行を実行するには、目的のデータベースに接続されているサーバーを起動します。新しいサーバーバージョンでデータベーススキーマが変更すると、データベースのレコード数が多すぎない限り、移行が自動的に開始されます。

たとえば、数百万件のレコードを含むテーブルにインデックスを作成すると、時間がかかり、サービスの大きな中断を引き起こす可能性があります。したがって、自動移行には **300000** レコードのしきい値が存在します。レコード数がこのしきい値を超えると、インデックスは作成されません。代わりに、手動で適用できる SQL コマンドを含む警告がサーバーログに表示されます。

しきい値を変更するには、デフォルトの **connections-liquibase** プロバイダーの値である **index-creation-threshold** プロパティーを設定します。

```
kc.[sh|bat] start --spi-connections-liquibase-default-index-creation-threshold=300000
```

3.3.2. 手動によるリレーショナルデータベース移行

データベーススキーマを手動でアップグレードするには、デフォルトの **connections-jpa** プロバイダーの **migration-strategy** プロパティー値を "manual" に設定します。

```
kc.[sh|bat] start --spi-connections-jpa-quarkus-migration-strategy=manual
```

この設定でサーバーを起動すると、サーバーはデータベースを移行する必要があるか確認します。必要な変更は **bin/keycloak-database-update.sql** SQL ファイルに書き込まれます。このファイルを確認してデータベースに対して手動で実行できます。

エクスポートされた SQL ファイルのパスと名前を変更するには、デフォルトの **connections-jpa** プロバイダーの **migration-export** プロパティーを設定します。

```
kc.[sh|bat] start --spi-connections-jpa-quarkus-migration-export=<path>/<file.sql>
```

このファイルをデータベースに適用する方法の詳細は、使用しているリレーショナルデータベースのドキュメントを参照してください。変更がファイルに書き込まれると、サーバーは終了します。

3.4. テーマの移行

カスタムテーマを作成した場合は、そのテーマを新しいサーバーに移行する必要があります。また、カスタマイズした部分によっては、組み込みテーマへの変更をカスタムテーマに反映することが必要になる場合があります。

手順

1. カスタムテーマを古いサーバーの **themes** ディレクトリーから新しいサーバーの **themes** ディレクトリーにコピーします。
2. 次のセクションを使用して、テンプレート、メッセージ、およびスタイルを移行します。
   - 移行の変更 に挙げられている更新されたテンプレートのいずれかをカスタマイズした場合は、ベーステーマのテンプレートと比較して、適用する必要のある変更があるかどうかを確認します。
   - メッセージをカスタマイズした場合は、キーまたは値を変更したり、追加のメッセージを追加したりする必要がある場合があります。

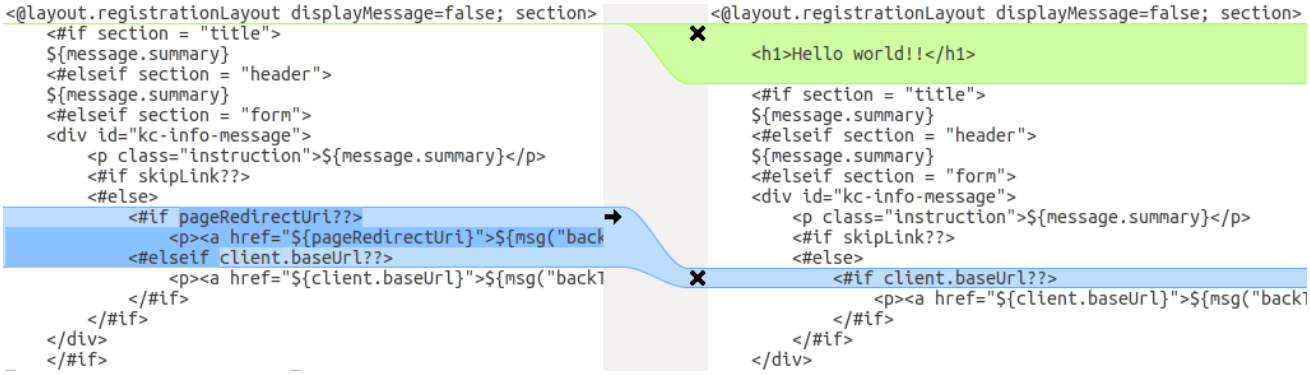
- スタイルをカスタマイズ済みで、Red Hat build of Keycloak のテーマを拡張する場合は、スタイルの変更を確認します。ベーステーマを拡張する場合は、この手順をスキップできます。

3.4.1. テンプレートの移行

テンプレートをカスタマイズした場合は、新しいバージョンを確認して、カスタマイズしたお使いのテンプレートを更新するかどうかを決定します。軽微な変更を加えた場合は、更新されたテンプレートとカスタマイズしたテンプレートを比較することを推奨します。ただし、多くの変更を加えた場合は、新しいテンプレートとお使いのカスタマイズしたテンプレートを比較することを検討してください。この比較により、どのような変更を加える必要があるかがわかります。

差分ツールを使用してテンプレートを比較できます。次のスクリーンショットは、Login テーマの **info.ftl** テンプレートとカスタムテーマの例を比較したものです。

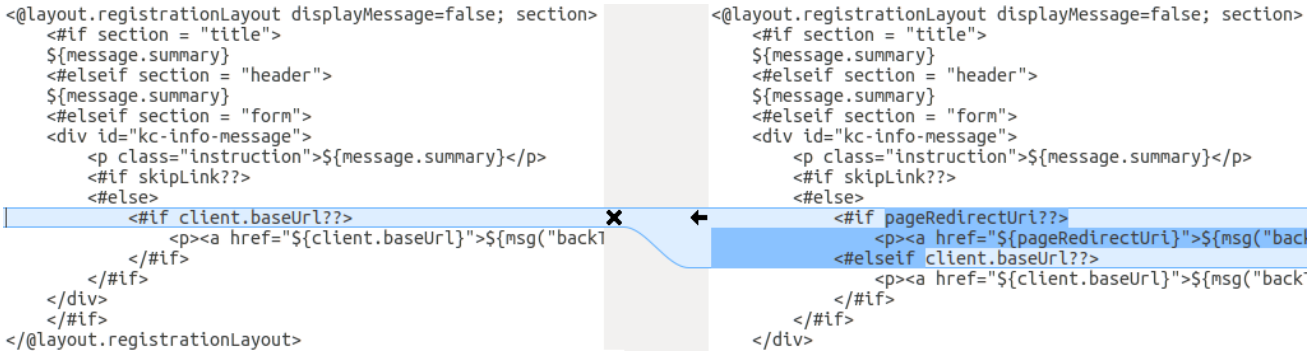
更新バージョンの Login テーマテンプレートとカスタムの Login テーマテンプレートの比較



この比較から、1 つ目の変更点 (**Hello world!!**) はカスタマイズであり、2 つ目の変更点 (**if pageRedirectUri**) はベーステーマの変更点であることがわかります。2 つ目の変更点をカスタムテンプレートにコピーすると、カスタマイズしたテンプレートが正常に更新されます。

別の方法として、次のスクリーンショットは、古いインストールの **info.ftl** テンプレートと新しいインストールの更新された **info.ftl** テンプレートを比較したものです。

古いインストールの Login テーマテンプレートと更新された Login テーマテンプレート



この比較から、ベーステンプレートの変更点がわかります。これをもとに、変更したテンプレートに同じ変更を手動で加えることができます。この方法はより複雑であるため、最初の方法が実行不可能な場合にのみ使用してください。

3.4.2. メッセージの移行

別の言語のサポートを追加した場合は、上記の変更点をすべて適用する必要があります。別の言語のサポートを追加していない場合は、何も変更する必要がない可能性があります。テーマ内の影響を受けるメッセージを変更した場合にのみ、変更を加える必要があります。

手順

1. 追加された値については、ベーステーマのメッセージ値を確認し、メッセージをカスタマイズする必要があるかどうかを判断します。
2. 名前が変更された鍵の場合は、カスタムテーマのキーの名前を変更します。
3. 変更された値については、ベーステーマの値を確認して、カスタムテーマに変更を加えなければならないかどうかを判断します。

3.4.3. スタイルの移行

組み込みテーマのスタイルに加えた変更を反映するには、カスタムスタイルを更新する必要がある場合があります。差分ツールを使用して、古いサーバーインストールと新しいサーバーインストール間のスタイルシートの変更を比較することを検討してください。

以下に例を示します。

```
$ diff RHSSO_HOME_OLD/themes/keycloak/login/resources/css/login.css \
RHSSO_HOME_NEW/themes/keycloak/login/resources/css/login.css
```

変更を確認し、それらがカスタムスタイルに影響するかどうかを判断します。

第4章 RED HAT BUILD OF KEYCLOAK アダプターのアップグレード

Red Hat build of Keycloak をアップグレードした後、アダプターをアップグレードできます。以前のバージョンのアダプターは、新しいバージョンの Red Hat build of Keycloak サーバーで動作する可能性がありますが、以前のバージョンの Red Hat build of Keycloak サーバーは、新しいバージョンのアダプターでは動作しない可能性があります。

4.1. 古いアダプターとの互換性

新しいバージョンの Red Hat build of Keycloak は、古いバージョンのアダプターで動作する可能性があります。ただし、Red Hat build of Keycloak サーバーの修正によって、古いバージョンのアダプターとの互換性が失われている可能性があります。たとえば、OpenID Connect 仕様の新しい実装が、古いクライアントアダプターのバージョンとマッチしない可能性があります。

このような状況では、互換モードを使用できます。OpenID Connect クライアントの場合は、管理コンソールのクライアント詳細ページに **OpenID Connect Compatibility Modes** が表示されます。このオプションを使用すると、古いクライアントアダプターとの互換性を維持するために、Red Hat build of Keycloak サーバーのいくつかの新しい機能を無効にできます。詳細は、個々のスイッチのツールヒントを参照してください。

4.2. EAP アダプターのアップグレード

JBoss EAP アダプターをアップグレードするには、次の手順を実行します。

手順

1. 新しいアダプターアーカイブをダウンロードします。
2. **EAP_HOME/modules/system/add-ons/keycloak/** ディレクトリーを削除して、以前のアダプターモジュールを削除します。
3. ダウンロードしたアーカイブを **EAP_HOME** に展開します。

4.3. JAVASCRIPT アダプターのアップグレード

Web アプリケーションにコピーされた JavaScript アダプターをアップグレードするには、以下の手順を実行します。

手順

1. 新しいアダプターアーカイブをダウンロードします。
2. ダウンロードしたアーカイブの **keycloak.js** ファイルで、アプリケーション内の **keycloak.js** ファイルを上書きします。

4.4. NODE.JS アダプターのアップグレード

Web アプリケーションにコピーした **Node.js** アダプターをアップグレードするには、次の手順を実行します。

手順

1. 新しいアダプターアーカイブをダウンロードします。
2. 既存の **Node.js** アダプターのディレクトリーを削除します。
3. 更新されたファイルをその場所に展開します。
4. アプリケーションの **package.json** で keycloak-connect の依存関係を変更します。

第5章 RED HAT BUILD OF KEYCLOAK 管理クライアントのアップグレード

admin-client をアップグレードする前に、必ず Red Hat build of Keycloak サーバーをアップグレードしてください。以前のバージョンの admin-client は、新しいバージョンの Red Hat build of Keycloak サーバーで動作する可能性がありますが、以前のバージョンの Red Hat build of Keycloak サーバーは、新しいバージョンの admin-client では動作しない可能性があります。したがって、Red Hat build of Keycloak サーバーと一致する admin-client バージョンを使用してください。