Red Hat build of Eclipse Vert.x 4.3

Eclipse Vert.x 4.3 リリースノート

Eclipse Vert.x 4.3.7 での使用

Last Updated: 2023-04-08

Red Hat build of Eclipse Vert.x 4.3 Eclipse Vert.x 4.3 リリースノート

Eclipse Vert.x 4.3.7 での使用

法律上の通知

概要

本リリースノートには、Eclipse Vert.x 4.3.7 に関連する重要な情報が含まれています。

目次

はじめに

リリース日: 2023-02-13

RED HAT ビルドの ECLIPSE VERT.X 4.3.7 - ライフサイクル終了の予定

Red Hat ビルドの Eclipse Vert.x 4.3.7 は、Red Hat が提供する予定の最後のサポート対象リリースです。フルサポートは 2023 年 5 月 31 日に終了します。詳細は、製品ライフサイクル ページを参照してください。Red Hat は、製品ライフサイクルが終了するまで、4.3.x リリースで Red Hat ビルドの Eclipse Vert.x のセキュリティーとバグ修正を提供し続けます。

Eclipse Vert.x アプリケーションは、Red Hat ビルドの Quarkus に移行できます。

Red Hat ビルドの Quarkus

Quarkus は、最適な Java ライブラリーと標準を使用して作成された、JVM およびネイティブコンパイル用に調整された Kubernetes ネイティブ Java フレームワークです。サーバーレス、マイクロサービス、コンテナー、Kubernetes、FaaS、クラウドなどの環境で Java アプリケーションを実行するための効果的なソリューションを提供します。
Quarkus のリアクティブ機能が Eclipse Vert.x を内部で使用しているため、Quarkus では Eclipse Vert.x アプリケーションを再利用できます。したがって、Eclipse Vert.x アプリケーションを Quarkus に移行することを推奨します。

詳細は、Quarkus の 製品ページ と ドキュメント を参照してください。

Red Hat は、移行作業を支援するためのリソースを作成する予定です。

RED HAT ドキュメントへのフィードバック (英語のみ)

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ている。

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. この問題を確認するには、フィードバックビューで問題トラッカーをクリックします。

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、Red Hat CTO である Chris Wright のメッセージ をご覧ください。

第1章 必要なインフラストラクチャーコンポーネントのバージョン

Red Hat ビルドの Eclipse Vert.x を使用する場合は、以下のコンポーネントを使用できます。ただし、Red Hat は、Red Hat OpenShift クラスターと Red Hat OpenJDK を除き、以下に示すコンポーネントに対するサポートは提供しません。

必要なコンポーネント

Eclipse Vert.x を使用してアプリケーションをビルドし、開発するには、以下のコンポーネントが必要です。

コンポーネント名	バージョン
Maven	3.6.0 以降
JDK[a]	8、11、または 17

[a] JRE はソースから Java アプリケーションをコンパイルするツールを提供しないため、完全な JDK インストールが必要です。

オプションのコンポーネント

Red Hat は、開発環境および実稼働環境に応じて以下のコンポーネントを使用することを推奨します。

コンポーネント名	バージョン
git	2.0 以降
OpenShift Maven プラグイン	1.1.1
oc コマンドラインツール	3.11 以降[a]
Red Hat OpenShift クラスターへのアクセス[b]	3.11 以降

[a] CLI ツール **oc** のバージョンは、使用している OCP のバージョンに対応する必要があります。

[b] OpenShiftCluster は Red Hat によってサポートされています。

第2章 サポートされる ECLIPSE VERT.X RUNTIME コンポーネント設定および統合

以下のリソースは、Eclipse Vert.x と Red Hat 製品のサポートされる設定および統合を定義します。

- 実稼働環境で Eclipse Vert.x との統合でサポートされる技術の一覧は、Red Hat Runtimes Supported Configurations を参照してください。
- Eclipse Vert.x ランタイムアーティファクトおよびそのバージョンの一覧は、コンポーネントの詳細ページ を参照してください。

第3章 機能

3.1. 新機能および変更された機能

本項では、本リリースで導入された新機能を説明します。また、既存の機能の変更に関する情報も含まれます。

3.1.1. 4.3 リリースで導入された新機能または変更された機能

Eclipse Vert.x 4.3 では、次の新機能または変更された機能が提供されます。

3.1.1.1. Micrometer はメトリックタイプを JMX オブジェクト名に追加します

Eclipse Vert.x 4.3.4 以降では、Micrometer 1.9.3 へのアップグレードにより、Java Management Extensions (JMX) で Eclipse Vert.x Micrometer Metrics を使用する場合、オブジェクト名にメトリックタイプが含まれるようになりました。

この拡張機能は、**micrometer-registry-jmx** モジュールのユーザーにのみ関連します。

3.1.1.2. GraphQL Java 19 を使用する Eclipse Vert.x は、デフォルトでプラットフォームロケールを使用します

Eclipse Vert.x 4.3.3 以降、Eclipse Vert.x は GraphQL Java のバージョン 19 をサポートします。これは、GraphQL クエリー言語の Java サーバー実装です。GraphQL Java 19 を使用する場合、JVM でロケールを設定しない場合、GraphQL エンジンは、JVM がインストールされているプラットフォームのロケールである JVM デフォルトロケールを使用するようになりました。あるいは、別の値を使用するように JVM デフォルト **Locale** を設定するか、Eclipse Vert.x Web GraphQL ハンドラーを使用してカスタムロケールを設定することができます。

注記

Eclipse Vert.x 4.3.3 以降では、バージョン 18 の GraphQL Java もサポートされています。

3.1.1.3. jackson-databind 機能を使用するユーザーは、プロジェクトにこの依存関係を含める必要があります

Eclipse Vert.x 4.3.2 以降では、Jackson Databind ライブラリーを **vertx-web-openapi**、**vertx-auth-webauthn**、または **vertx-config-yaml** モジュールで使用する場合、次の依存関係をプロジェクト記述子に追加する必要があります。

```
<dependency>
  <groupId>com.fasterxml.jackson.core</groupId>
  <artifactId>jackson-databind</artifactId>
</dependency>
```

Jackson Databind ライブラリーはいくつかのセキュリティー上の脆弱性を引き起こし、他のモジュールは通常、何らかの内部動作を行うためにのみ Jackson Databind を使用するため、Eclipse Vert.x パーサーの使用は、Jackson Databind とともに **vertx-web-openapi**、**vertx-auth-webauthn**、または **vertx-config-yaml** モジュールを使用する必要よりも優先されます。ただし、Jackson Databind でこれらのモジュールのいずれかを引き続き使用する場合は、前の例に示すように、プロジェクトにこの依存関係を明示的に含める必要があります。

3.1.1.4. Eclipse Vert.x OpenAPI でのボディハンドラー設定の変更点

Eclipse Vert.x 4.3.1 以降、Eclipse Vert.x OpenAPI では **routerBuilder.rootHandler()** メソッドを使用して、PLATFORM または SECURITY_POLICY ハンドラーの **後** にボディハンドラーが正しい順序で設定されるようにする必要があります。

以下に例を示します。

```
BodyHandler bodyHandler = BodyHandler.create("my-uploads");
routerBuilder.rootHandler(bodyHandler);
```

Eclipse Vert.x の以前のリリースでは、Eclipse Vert.x OpenAPI は、本体ハンドラーを追加するための **routerBuild.bodyHandler()** メソッドをサポートしていました。ただし、**bodyHandler()** メソッドには次の欠点がありました。

- Eclipse Vert.x は、セットアップが正しい順序で行われたことを確認するための検証を実行しませんでした。
- Eclipse Vert.x OpenAPI は、body ハンドラーを特別なハンドラーとして格納して、常にルートの最初のハンドラーになるようにしましたが、常に保証されるわけではありませんでした。

bodyHandler() メソッドは、Eclipse Vert.x 4.3.1 で非推奨になりました。以前の **rootHandler** 呼び出しは、以前のバージョンで使用可能だった次の **bodyHandler** 呼び出しに取って代わります。

```
BodyHandler bodyHandler = BodyHandler.create("my-uploads");
routerBuilder.bodyHandler(bodyHandler);
```

3.1.1.5. BLOB および RAW データ値に対する Eclipse Vert.x リアクティブ Oracle クライアントの機能強化

Eclipse Vert.x 4.3.1 以降、Eclipse Vert.x リアクティブ Oracle クライアントには、**BLOB** および **RAW** データに対する次の機能強化が含まれています。

- **BLOB** または **RAW** データを読み取る場合、クライアントは **io.vertx.core.buffer.Buffer** 値を返すようになりました。
  以下に例を示します。

```
client.preparedQuery("SELECT data FROM images WHERE id = ?")
    .execute(Tuple.of(id))
    .onComplete(ar -> {
        if (ar.succeeded()) {
            Row row = ar.result().iterator().next();

            // Use io.vertx.core.buffer.Buffer when reading
            Buffer data = row.getBuffer("data");
        }
    });
```

注記

この変更は、クエリーパラメーターとしての **RAW** 値に関する問題を修正する一環として、一貫性を保つために導入されました。Eclipse Vert.x の以前のリリースでは、**BLOB** または **RAW** データはバイト配列として返されました。

- **BLOB** データの書き込みまたはフィルタリング時に、データが新しい **io.vertx.oracleclient.data.Blob** 型で表されるようになりました。
以下に例を示します。

```
client.preparedQuery("INSERT INTO images (name, data) VALUES (?, ?)")
  // Use io.vertx.oracleclient.data.Blob when inserting
  .execute(Tuple.of("beautiful-sunset.jpg", Blob.copy(imageBuffer)))
  .onComplete(ar -> {
    // Do something
  });
```

3.1.1.6. 自動生成されたキーの取得はデフォルトで無効になっています

Eclipse Vert.x 4.3.1 以降、Eclipse Vert.x Oracle リアクティブクライアントでは、自動生成されたキーの取得はデフォルトで無効になっています。ほとんどのアプリケーションは **ROWID** に依存しないため、Eclipse Vert.x Oracle リアクティブクライアントは通常、自動生成されたキーを取得する必要はありません。

この機能強化により、自動生成されたキーの取得が有効になっていると正常に実行できない **INSERT… SELECT** などのクエリーも容易になります。

3.1.1.7. io.vertx.core.shareddata.ClusterSerializable インターフェイスの使用

Eclipse Vert.x 4.3.0 以降、Eclipse Vert.x は **io.vertx.core.shareddata.ClusterSerializable** インターフェイスをサポートし、これらのオブジェクトが AsyncMap から読み取られるか、またはバッファーからデコードされるときに、バッファーとの間でオブジェクトを読み書きします。EventBus メッセージ本文。

Eclipse Vert.x の以前のリリースでは、**io.vertx.core.shareddata.impl.ClusterSerializable** インターフェイスがサポートされていました。ただし、このインターフェイスは実装パッケージで提供されていたため、信頼性が低い可能性があると考えられていました。**io.vertx.core.shareddata.impl.ClusterSerializable** インターフェイスは、Eclipse Vert.x 4.3.0 で廃止され、公開されました。

3.1.1.8. Micrometer リクエストメトリックの requestsTagsProvider オプションの名前変更

Eclipse Vert.x 4.3.0 以降、**MicrometerMetricsOptions** クラスで、サーバー要求メトリックの **requestsTagsProvider** オプションの名前が **serverRequestTagsProvider** に変更されました。同様の **clientRequestTagsProvider** オプションがクライアント要求メトリックにも使用できるようになったため、この拡張が必要です。

Eclipse Vert.x の以前のリリースでは、**requestsTagsProvider** オプションは、それぞれ **getRequestsTagsProvider** と **setRequestsTagsProvider** という名前の getter と setter を使用していました。Eclipse Vert.x 4.3.0 以降のバージョンでは、**serverRequestTagsProvider** オプションのゲッターとセッターの名前が **getServerRequestTagsProvider** と **setServerRequestTagsProvider** に変更されました。

3.1.1.9. OAuth2 OBO 呼び出しは、 TokenCredentials ではなく明示的な OAuth2Credentials を想定しています

Eclipse Vert.x 4.3.0 以降では、OAuth2 認証が on-behalf-of (OBO) モードで設定されている場合、OAuth2 では、リクエストを認証するために **OAuth2Credentials** オブジェクトを明示的に指定する必要があります。

以下に例を示します。

```
oauth2.authenticate(
  new Oauth2Credentials().setAssertion("head.body.signature").addScope("a").addScope("b"))
```

Eclipse Vert.x の以前のリリースでは、OBO モードの OAuth2 認証で **TokenCredentials** を使用できました。ただし、このフローはオプションであるため、同じ **OAuth2Credentials** オブジェクトを再利用できるようにするために、以前のバージョンで使用できた次のタイプの **TokenCredentials** 呼び出しが、前述の **Oauth2Credentials** 呼び出しに取って代わります。

```
oauth2.authenticate(
  new TokenCredentials("head.body.signature").addScope("a").addScope("b"));
```

3.1.1.10. RoutingContext.fileUploads() メソッドはリストを返します

Eclipse Vert.x 4.3.0 以降では、**RoutingContext.fileUploads()** メソッドは **List<FileUpload>** 値を返します。ファイルのアップロードをリストに保存すると、アップロードの順序を維持するのに役立ちます。

以下に例を示します。

```
List<FileUpload> uploads = ctx.fileUploads();
```

Eclipse Vert.x の以前のリリースでは、**RoutingContext.fileUploads()** メソッドは **Set<FileUpload>** 値を返しました。ただし、ファイルアップロードをセットに格納することは、フォームコンテンツタイプの World Wide Web Consortium (W3C) 仕様と一致しませんでした。これは、正しい順序が保持されず、ユーザーがアップロード名を一意のキーとして信頼できないためです。前の例は、以前のバージョンで使用できた次のメソッド宣言に取って代わります。

```
Set<FileUpload> uploads = ctx.fileUploads();
```

3.1.1.11. サブルーターを実装する単一の方法

Eclipse Vert.x 4.3.0 以降では、**Route.subRouter(Router)** メソッドが、サブルーターを実装するためにサポートされている唯一の方法です。

Eclipse Vert.x の以前のリリースでは、サブルーターを実装するための 2 つの異なる方法がサポートされていました。

- **Route.subRouter(Router)**
- **Router.mountSubRouter(String, Router)**

ただし、**Router.mountSubRouter** メソッドは任意のパスを許可するのに対し、**Route.subRouter** メソッドはサブルーティングパスを表すためにワイルドカードアスタリスク (*) を明示的に要求するため、これら 2 つのメソッド間の動作には一貫性がありませんでした。**Router.mountSubRouter** メソッドも、欠落しているワイルドカードを追加することで **Route.subRouter** メソッドに委任されました。

Eclipse Vert.x 4.3.0 以降のバージョンでは、**Router.mountSubRouter** メソッドは非推奨です。Router オブジェクトは、サブルーターを実装するために **Route.subRouter** メソッドも使用する必要があります。以下に例を示します。

```
router.route("/eventbus/*").subRouter(otherRouter);
```

以前の **router.route().subRouter()** 呼び出しは、以前のバージョンで使用可能だった次のタイプの **router.mountSubRouter()** 呼び出しに取って代わります。

```
router.mountSubRouter("/eventbus", otherRouter);
```

注記

以前のリリースでは、ルーターオブジェクトは、**router.mountSubRouter()** を使用する代わりに、**router.route().subRouter()** 呼び出しを使用することもできました。

3.1.1.12. 複数のハンドラー呼び出しにわたる解析済み要求本文のキャッシュ

Eclipse Vert.x 4.3.0 以降では、ボディハンドラーが Web リクエストのボディを解析した後、ボディハンドラーはボディバッファーをリクエストのルーティングコンテキストに提供します。 ルーティングコンテキストでボディバッファーをキャッシュすることは、要求ボディのデコードされたビューを必要とする複数の異なるハンドラーが、ボディを再度解析することなく、キャッシュされた結果を取得できることを意味します。 この拡張機能は、本文のコンテンツがタイプ **application/json** である状況もサポートします。

RoutingContext クラスは、要求本文を指定されたタイプとして取得するために使用される新しい **body()** メソッドを提供します。

以下に例を示します。

```
RoutingContext.body().asString()
RoutingContext.body().asString(String encoding)
RoutingContext.body().asJsonObject()
RoutingContext.body().asJsonArray()
RoutingContext.body().asJsonObject(int maxLength)
RoutingContext.body().asJsonArray(int maxLength)
RoutingContext.body().buffer()
```

新しい **body()** ゲッターは、次の追加機能も提供します。

```
// the length of the buffer (-1) for null buffers
RoutingContext.body().length()

// Converting to POJO
RoutingContext.body().asPOJO(Class<T> clazz)
RoutingContext.body().asPOJO(Class<T> clazz, int maxLength)
```

この拡張機能には、次の利点があります。

- **body()** ゲッターが null になることはありません。これにより、null チェックを実行する必要がなくなります。
- ベースバッファーが変更されない限り、要求本文は 1 回だけ解析する必要があります。ベースバッファーが変更されると、別の解析がトリガーされ、キャッシュされた値はその時点でオーバーライドされます。

Eclipse Vert.x の以前のリリースでは、**RoutingContext** クラスは次のメソッドを提供していましたが、現在は **body()** メソッドを使用するために推奨されていません。

```
RoutingContext.getBodyAsString()
```

```
RoutingContext.getBodyAsString(String encoding)
RoutingContext.getBodyAsJson()
RoutingContext.getBodyAsJsonArray()
RoutingContext.getBodyAsJson(int maxLength)
RoutingContext.getBodyAsJsonArray(int maxLength)
RoutingContext.getBody()
```

3.1.1.13. EventBus 通知のデフォルトの変更

Eclipse Vert.x 4.3.0 以降では、不必要なトラフィックを回避するために、Eclipse Vert.x サーキットブレーカーの状態の変化に関する通知がデフォルトで無効になっています。これらの通知を有効にするには、null 以外のパラメーターを指定して **CircuitBreakerOptions** オブジェクトの **setNotificationAddress** メソッドを呼び出します。

以下に例を示します。

```
CircuitBreakerOptions options = new CircuitBreakerOptions()
    .setNotificationAddress(CircuitBreakerOptions.DEFAULT_NOTIFICATION_ADDRESS);
```

前の例に示すように通知を有効にすると、既定の動作では、ローカルコンシューマーにのみ通知が送信されます。クラスター全体で通知を送信するには、**false** のパラメーターを指定して **setNotificationLocalOnly** メソッドを呼び出します。

以下に例を示します。

```
CircuitBreakerOptions options = new CircuitBreakerOptions()
    .setNotificationAddress(CircuitBreakerOptions.DEFAULT_NOTIFICATION_ADDRESS)
    .setNotificationLocalOnly(false);
```

3.1.1.14. MySQL クライアントのバッチ実行の変更

Eclipse Vert.x 4.3.0 以降、Eclipse Vert.x リアクティブ SQL クライアントはパイプラインクエリーをサポートし、デフォルトでパイプラインモードでバッチクエリーを実行します。パイプライン処理とは、前の要求への応答を待たずに、同じ接続で要求が送信されることを意味します。

Eclipse Vert.x の以前のリリースでは、MySQL にはバッチ処理のネイティブプロトコルサポートがないため、SQL クライアントは準備されたクエリーを順番に実行することでバッチクエリーを実行し、ユーザーは API 呼び出しを介して直接操作できました。

3.1.1.15. ヒントとヒント文字列の MongoDB の機能強化

Eclipse Vert.x 4.3.0 以降、Eclipse Vert.x には、ヒントとヒント文字列に関する次の MongoDB 拡張機能が含まれています。

- **FindOptions** オブジェクトは、タイプ **JSONObject** のヒントをサポートするようになりました。これは、**FindOptions** オブジェクトが **String** 型のヒントをサポートしていた以前のリリースの動作に取って代わります。
- **BulkOperations** および **UpdateOptions** オブジェクトは、タイプ **JSONObject** のヒントもサポートするようになりました。**BulkOperations** および **UpdateOptions** クラスはそれぞれ、この目的のために **getHint()** および **setHint()** メソッドを提供します。
- **BulkOperations**、**UpdateOptions**、および **FindOptions** オブジェクトも、**String** 型のヒント文字列をサポートするようになりました。**BulkOperations**、**UpdateOptions**、および

FindOptions クラスはそれぞれ、この目的のために **getHintString ()** および **setHintString()** メソッドを提供します。

3.1.1.16. スキーマの構築における変更

Eclipse Vert.x 4.3.0 以降では、スキーマを構築するときに、Eclipse Vert.x JSON スキーマが提供する JSON 表現を使用します。JSON 表現では、任意のバリデーターを使用できます。

以下に例を示します。

```
JsonSchema schema = JsonSchema.of(dsl.toJson());
```

Eclipse Vert.x の以前のリリースでは、**SchemaBuilder** クラスが **build()** メソッドを提供していましたが、これにはバリデーターの特定の実装を使用する必要がありました。**build()** メソッドは、Eclipse Vert.x 4.3.0 で非推奨になりました。前述の JsonSchema の例は、以前のバージョンで使用できた次のタイプの **build()** メソッド呼び出しに取って代わります。

```
Schema schema = dsl.build(parser);
```

3.1.2. 以前の 4.x リリースで導入された新機能

以下の新機能は、以前の 4.x リリースで導入されました。

3.1.2.1. Java 17 のサポート

Eclipse Vert.x 4.2.7 以降、Eclipse Vert.x は Red Hat OpenJDK 17 での使用が認定されています。

3.1.2.2. RequestOptions での HTTP ヘッダーの検証

Eclipse Vert.x 4.2.4 以降では、**RequestOptions** メソッドは HTTP ヘッダーを検証し、ヘッダー名が無効な場合に要求が失敗します。

Eclipse Vert.x の以前のリリースでは、**RequestOptions** メソッドではヘッダー名を検証しない Multimap 実装が使用されていたため、**HTTPClientRequest** は HTTP ヘッダーを検証していました。

3.1.2.3. 照合用のデフォルトロケールとしての simple の使用

Eclipse Vert.x 4.2.4 以降では、**simple** ロケールが MongoDB collation のデフォルトロケールとして使用されます。

Eclipse Vert.x 4.2.3 では、文字列を比較するための言語固有のルールをサポートする照合オプションに対するサポートが導入されました。Eclipse Vert.x 4.2.3 では、プラットフォームのデフォルトがデフォルトのロケールとして使用されていました。ただし、プラットフォームのデフォルトは定数値ではないため、MongoDB でサポートされていないロケールを使用するシステムでエラーが発生する可能性がありました。たとえば、**Locale.FR** は正常に機能しますが、**Locale.FR_FR** はサポートされません。

3.1.2.4. StaticHandler ファイルシステム設定の変更

Eclipse Vert.x 4.2.4 以降では、webroot ディレクトリーおよびファイルシステムアクセスの **StaticHandler** 設定プロパティーは、**StaticHandler** ファクトリーコンストラクター呼び出しで定義されます。

たとえば、以下のコンストラクター呼び出しは、webroot ディレクトリー、**static/resources**、および相対ファイルシステムアクセスを定義します。

```
StaticHandler.create(FileSystemAccess.RELATIVE, "static/resources");
```

たとえば、以下のコンストラクター呼び出しは、webroot ディレクトリー、**/home/paulo/Public**、および root ファイルシステムのアクセスを定義します。

```
StaticHandler.create(FileSystemAccess.ROOT, "/home/paulo/Public");
```

Eclipse Vert.x の以前のリリースでは、**allowRootFileSystemAccess** および **webroot** プロパティーはセッターを使用して定義されていました。ただし、このプロパティーの値は最終的なものではなかったため、無効な静的設定が生じる可能性がありました。Eclipse Vert.x 4.2.4 では、前述のコンストラクター呼び出しは以下のセッター宣言に優先されるようになりました。

```
StaticHandler.create()
  .setAllowRootFileSystemAccess(true)
  .setWebRoot("/home/paulo/Public");
```

注記

StaticHandler.create() メソッドは、以前のリリースと同様に **RELATIVE** および **webroot** のデフォルト値を使用します。

3.1.2.5. verticle 内でのランダムサーバーポートの共有

Eclipse Vert.x 4.2.0 以降では、**-1** などの負のポート番号にバインドされた 2 つの異なる HTTP サーバーは、特定の verticle デプロイメントのインスタンス内で同じランダムポートを共有します。つまり、ポート **-1** にバインドされている複数の HTTP サーバーが、同じランダムポートを共有します。同様に、ポート **-2** にバインドされている複数の HTTP サーバーは、同じランダムポートを共有します。異なる HTTP サーバーが異なるランダムポートを持つことが可能になるため、負のポート番号に基づくこのポート共有の動作は verticle とは無関係です。

Eclipse Vert.x の以前のリリースでは、ランダムサーバーポート共有は、ポート **0** にバインドされた 2 つの HTTP サーバーに基づいていました。しかし、これにより、同じ verticle のインスタンス内に、異なるランダムポートを持つ 2 つの HTTP サーバーをバインドできなくなりました。

3.1.2.6. HTTP サーバークッキーの変更

Eclipse Vert.x 4.2.0 には、すべてのクッキーを取得できる新しいメソッド **Set<Cookie> cookies()** が含まれます。

Vert.x の以前のリリースでは、**HttpServerRequest** および **HttpServerResponse** インターフェイスは、EclipseVert.x4.2.0 で非推奨となった次のメソッドを使用していました。

```
Map<String, Cookie> cookieMap()
```

RFC 6265 - HTTP 状態管理メカニズム の仕様では、各クッキーはタプル **<name, domain, path>** に基づいて一意に識別されると規定されています。しかし、EclipseVert.x の以前のリリースで使用されていた **Map<String, Cookie> cookieMap()** メソッドは、クッキーが名前のみに基づいて識別できると誤って想定していました。つまり、複数のクッキーが同じ名前を共有している場合、マップは解析される最後のクッキーを保持し、以前に解析された値はすべて警告なしで上書きされていました。

3.1.2.7. GraphQLContext オブジェクトによるコンテキスト管理

Eclipse Vert.x 4.2.0 は、GraphQL クエリー言語の Java サーバー実装である GraphQL Java のバージョン 17 をサポートします。GraphQL Java 17 では、**GraphQLContext** オブジェクトが GraphQL Java アプリケーションのコンポーネント間でコンテキストデータを共有するための標準になりました。

Eclipse Vert.x 4.2.0 では、GraphQL の実行を設定するために以下の新しいメカニズムが導入されました。

```
GraphQLHandler handler = GraphQLHandler.create(graphQL).beforeExecute(builderWithContext ->
{
  DataLoader<String, Link> linkDataLoader = DataLoaderFactory.newDataLoader(linksBatchLoader);
  DataLoaderRegistry dataLoaderRegistry = new DataLoaderRegistry().register("link",
linkDataLoader);
  builderWithContext.builder().dataLoaderRegistry(dataLoaderRegistry);
});
```

Eclipse Vert.x の以前のリリースでは、データローダーを設定するために Vert.x Web GraphQL ハンドラーで次のフックが使用されていました。以下のフックは Eclipse Vert.x 4.2.0 で非推奨になりました。

```
GraphQLHandler handler = GraphQLHandler.create(graphQL).dataLoaderRegistry(rc -> {
  DataLoader<String, Link> linkDataLoader = DataLoader.newDataLoader(linksBatchLoader);
  return new DataLoaderRegistry().register("link", linkDataLoader);
});
```

3.1.2.8. OpenJDK11 OpenShift イメージは複数のアーキテクチャーをサポートします

IBM Z および IBMPowerSystems の OpenJ9 イメージは非推奨になりました。次の OpenJDK11 イメージは、複数のアーキテクチャーをサポートするように更新されています。

- **ubi8/openjdk-11**

OpenJDK11 イメージは、次のアーキテクチャーで使用できます。

- x86 (x86_64)
- s390x (IBM Z)
- ppc64le (IBM Power Systems)

3.1.2.9. FIPS 対応の Red Hat Enterprise Linux (RHEL) システムでの Eclipse Vert.x Runtime のサポート

Eclipse Vert.x の Red Hat ビルドは、FIPS 対応の RHEL システムで実行され、RHEL が提供する FIPS 認定ライブラリーを使用します。

3.1.2.10. HTTP クライアントのリダイレクトハンドラーのヘッダー伝播

Eclipse Vert.x 4.1.0 以降では、HTTP リダイレクトにヘッダーがある場合は、HTTP クライアントのリダイレクトハンドラーはヘッダーを次の要求に伝播します。この変更により、リダイレクトハンドラーは、リダイレクトされた要求全体をより詳細に制御できるようになります。

以前のリリースの Eclipse Vert.x では、ヘッダーのあるリダイレクト要求があったため、HTTP クライアントはリダイレクト後にヘッダーを処理していました。

以下の例は、Eclipse Vert.x 4.1.0 でリダイレクトを処理する方法を示しています。

```
RequestOptions options = new RequestOptions();
options.setMethod(HttpMethod.GET);
options.setHost(uri.getHost());
options.setPort(port);
options.setSsl(ssl);
options.setURI(requestURI);

// From 4.1.0 propagate headers
options.setHeaders(resp.request().headers());
options.removeHeader(CONTENT_LENGTH);
```

3.1.2.11. Infinispan 12 へのアップグレード

Eclipse Vert.x 4.1.0 では、Infinispan クラスターマネージャーが更新され、Infinispan 12 をベースにしています。

Infinispan 11 には、マルチマップキャッシュにバイトアレイを保存できないバグがありました。Eclipse Vert.x クラスターマネージャーは、内部 Infinispan クラス **WrappedBytes** を使用して eventbus サブスクリプションデータを保存する必要がありました。この問題は Infinispan 12 で修正されました。

3.1.2.12. MongoDB クライアントの接続文字列オプションより JSON 設定を優先

Eclipse Vert.x 4.1.0 では、**connection_string** オプションが利用可能な場合でも JSON 設定オプションが適用されます。

次の設定オプションが適用されるようになりました。

```
{
   mongo:{
      db_name: "mydb"
      connection_string: "mongodb://localhost:27017"
      maxPoolSize: 10
      minPoolSize: 3
   }
}
```

以前のリリースの Eclipse Vert.x では、接続文字列が利用可能な場合に JSON 設定オプションは無視されていました。たとえば、前述の例を見てみましょう。以前のリリースの Eclipse Vert.x では、**db_name**、**maxPoolSize**、および **minPoolSize** オプションは無視されていました。

3.1.2.13. 非推奨の JWT オプションメソッドの削除

Eclipse Vert.x 4.0 以降では、スコープの処理に JWT および OAuth2 ハンドラーが使用されます。

Eclipse Vert.x 4.1.0 以降、**JWTOptions.setScopes(List<String>)**、**JWTOptions.addScope(String)**、および **JWTOptions.withScopeDelimiter(String)** メソッドが削除されました。これらのメソッドは仕様に準拠していませんでした。

以下の例は、Eclipse Vert.x 4.1.0 でスコープを処理する方法を示しています。

```
// before 4.1.0
JWTAuthOptions authConfig = new JWTAuthOptions()
```

```
  .setJWTOptions(new JWTOptions()
    .addScope("a")
    .addScope("b")
    .withScopeDelimiter(" ")));

JWTAuth authProvider = JWTAuth.create(vertx, authConfig);

router.route("/protected/*").handler(JWTAuthHandler.create(authProvider));

// in 4.1.0
JWTAuth authProvider = JWTAuth.create(vertx, new JWTAuthOptions());

router.route("/protected/*").handler(
  JWTAuthHandler.create(authProvider)
    .addScope("a")
    .addScope("b")
    .withScopeDelimiter(" "));
```

3.1.2.14. 関数を受け入れるカスタムフォーマッターメソッドの非推奨

Eclipse Vert.x 4.1.0 から **LoggerHandler.customFormatter(Function)** メソッドが非推奨になりました。この関数は **HttpServerRequest** を入力として取り、フォーマットされたログ文字列を返します。出力は文字列であるため、コンテキストにアクセスすることはできません。

代わりに新しいメソッド **LoggerHandler customFormatter(LoggerFormatter formatter)** を使用してください。このメソッドは、コンテキストへのアクセスを提供するカスタムフォーマッターを入力として取ります。

3.1.2.15. HTTP の失敗を処理する新しい例外

Eclipse Vert.x 4.1.0 以降では、HTTP の失敗の処理に使用できる新しい例外クラス **io.vertx.ext.web.handler.HttpException** を利用できます。この例外を使用して、500 以外のカスタムステータスコードを指定できます。たとえば、新規の **HttpException(401, "Forbidden")** は、禁止されているリクエストがステータスコード 401 を返す必要があることを示します。

3.1.2.16. RxJava 3 のサポート

Eclipse Vert.x 4.1.0 以降では、RxJava 3 がサポートされます。

- 新しい rxified API が **io.vertx.rxjava3** パッケージで利用できます。
- Eclipse Vert.x JUnit5 との統合は、**vertx -junit5-rx-java3** バインディングによって提供されます。

3.1.2.17. すべてのタイプのデータをバインドしてより安全になったコンテキストサーバーインターセプター

Eclipse Vert.x 4.0.3 以降、**ContextServerInterceptor.bind ()** メソッドはすべてのタイプのデータをコンテキストにバインドします。このメソッドはストレージの詳細を公開しないため、より安全になりました。

Eclipse Vert.x 4.0.3 より前のリリースでは、このメソッドは 'String' データ型のみをコンテキストにバインドしていました。また、ストレージの詳細も公開しました。

更新された **ContextServerInterceptor.bind ()** メソッドを使用するには、アプリケーションを更新する必要があります。

以下の例は、Eclipse Vert.x 4.0.3 より前のリリースのコードを示しています。

```
// Example code from previous releases

class X extends ContextServerInterceptor {
  @Override
  public void bind(Metadata metadata, ConcurrentMap<String, String> context) {
```

次の例は、Eclipse Vert.x 4.0.3 リリースの置換コードを示しています。

```
// Replacing code for Eclipse Vert.x 4.0.3 release

class X extends ContextServerInterceptor {
  @Override
  public void bind(Metadata metadata) {
```

3.1.2.18. ワイルドカード文字で終わるルートパスの末尾のスラッシュ (/) の一致が不要

Eclipse Vert.x 4.0.3 より前のリリースでは、ルートがスラッシュで終わるパスとワイルドカード /* で定義されている場合、一致するリクエストにも末尾のスラッシュ / が含まれている場合にのみ、ルートが呼び出されました。このルールは、ワイルドカードが空の場合に問題を引き起こしました。

Eclipse Vert.x 4.0.3 以降では、このルールは適用されなくなりました。パスがスラッシュ (/) で終わるルートを作成できます。ただし、リクエスト URL にスラッシュを指定することは必須ではありません。

また、リクエスト URL を作成および使用し、パスにスラッシュ (/) ではなく、ワイルドカードで終わるルートを呼び出すこともできます。たとえば、ワイルドカードが含まれるルートは **/foo/*** として定義できます。ここでは、ルートはパスの最後にあるオープンワイルドカードと一致する必要があります。リクエスト URL は **/foo** にすることができます。

この表は、Eclipse Vert.x 4.0.3 以前のリリースでのリクエスト URL **/foo/** * を送信するときの動作を示しています。Eclipse Vert.x 4.0.3 では終了スラッシュが任意であり、要求はルートに一致することがわかります。

ルート	Eclipse Vert.x 4.0.3	Eclipse Vert.x 4.0.3 より前のリリース
/foo	Match	No Match
/foofighters	No Match	No Match
/foo/	Match	Match
/foo/bar	Match	Match

3.1.2.19. サービス検出オプションから autoRegistrationOfImporters 属性を削除

AutoRegistrationOfImporters 属性はサービス検出オプションから削除されました。

3.1.2.20. 入力クレデンシャルとして token をサポートするように認証プロバイダークラスの認証メソッドを更新

Eclipse Vert.x 4.0.3 より前のリリースでは、AuthenticationProvider.authenticate () メソッドが入力クレデンシャルとして **jwt: someValue** を誤って取得していました。

Eclipse Vert.x 4.0.3 以降、AuthenticationProvider.authenticate() メソッドが更新され、**token: someValue** が入力クレデンシャルとして取得されます。この変更により、JSON API と型指定された API の両方が一貫性を保ち、同じ意味で使用できるようになります。

以下のコードは、Eclipse Vert.x 4.0.3 より前のリリースでの認証メソッドの実装を示しています。

```
new JsonObject().put("jwt", "token...");
```

以下のコードは、Eclipse Vert.x 4.0.3 リリースの認証メソッドの実装を示しています。

```
new JsonObject().put("token", "token...");
```

3.1.2.21. PEM キーの Get メソッドが String ではなく Buffer を返す

PubSecKeyOptions.getBuffer() メソッドは、PEM またはシークレットキーバッファーを返します。Eclipse Vert.x 4.0.2 より前のリリースでは、キーバッファーが保存され **String** として返されました。ただし、シークレットを **Buffer** として保存することを推奨します。Eclipse Vert.x 4.0.2 以降では、メソッドはキーバッファーを保存し、**Buffer** として保存し返します。この変更により、シークレットのセキュリティーおよび処理が改善します。

PubSecKeyOptions.setBuffer() メソッドは **String** 引数を受け入れます。set メソッドでは、ASCII 以外のシークレット資料を安全に処理するために、バッファーのオーバーロードが追加されました。この変更には、既存のコードを変更する必要はありません。

3.1.2.22. Kubernetes サービスインポーターが自動的に登録されなくなる

Eclipse Vert.x 4 以降、**KubernetesServiceImporter** 検出ブリッジは自動的に登録されなくなりました。Maven プロジェクトのクラスパスにブリッジを追加しても、自動的に登録されません。

ServiceDiscovery インスタンスの作成後にブリッジを手動で登録する必要があります。

3.1.2.23. 非同期操作に future メソッドを使用

Eclipse Vert.x 4 は、非同期操作に future を使用します。すべての callback メソッドには、対応する future メソッドがあります。

future は非同期操作の作成に使用できます。future を使用する場合は、エラー処理の方が優れています。したがって、アプリケーションでコールバックと future の組み合わせを使用することが推奨されます。

3.1.2.24. Jackson Databind ライブラリーの依存関係がない

Eclipse Vert.x 4 では、Jackson Databind は任意の Maven 依存関係です。この依存関係を使用する場合は、クラスパスに明示的に追加する必要があります。たとえば、オブジェクトマッピング JSON の場合は、依存関係を明示的に追加する必要があります。

3.1.2.25. 非推奨と削除の処理

Eclipse Vert.x 4 では、新機能が追加されています。以前の機能および機能は Eclipse Vert.x 4 で非推奨または削除されました。アプリケーションを Eclipse Vert.x 4 に移行する前に、非推奨および削除を確認します。

Java コンパイラーは、非推奨の API が使用されたときに警告を生成します。アプリケーションを Eclipse Vert.x 4 に移行する際に、コンパイラーを使用して非推奨のメソッドを確認できます。

3.1.2.26. 分散トレースのサポート

Eclipse Vert.x 4 は分散トレースをサポートします。トレースを使用してマイクロサービスを監視し、パフォーマンスの問題を特定することができます。

Eclipse Vert.x 4 は OpenTracing システムと統合します。

以下の Eclipse Vert.x コンポーネントはトレースをログに記録できます。

- HTTP サーバーおよび HTTP クライアント
- Eclipse Vert.x SQL クライアント
- Eclipse Vert.x Kafka クライアント

重要

トレースはテクノロジープレビューとして利用できます。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。テクノロジープレビューの機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行いフィードバックを提供していただくことを目的としています。

テクノロジープレビュー機能のサポート範囲は、Red Hat カスタマーポータルの テクノロジープレビュー機能のサポート範囲 を参照してください。

3.1.2.27. EventBus JavaScript Client の新しい公開場所

Eclipse Vert.x 4 では、EventBus JavaScript クライアント **vertx-web-client.js** は Maven リポジトリーの Red Hat アーティファクトとして公開されません。

クライアントは npm リポジトリーに公開されます。@vertx/eventbus-bridge-client.js からクライアントにアクセスできます。

3.1.2.28. OpenShift Maven プラグインを使用した Eclipse Vert.x アプリケーションのデプロイ

OpenShift Maven プラグインを使用して、OpenShift に Eclipse Vert.x アプリケーションをデプロイします。Fabric8 Maven プラグインはサポート対象外になりました。詳細は、migrating from Fabric8 Maven Plugin to Eclipse JKube セクションを参照してください。

3.1.2.29. OpenShift の Eclipse Vert.x メータリングラベル

メータリングラベルを Eclipse Vert.x Pod に追加し、OpenShift Metering Operator で Red Hat サブスクリプションの詳細を確認できます。

注記

- メータリングラベルは、Operator またはテンプレートがデプロイおよび管理する Pod に追加しないでください。
- OpenShift Container Platform バージョン 4.8 以前では、Metering Operator を使用してラベルを Pod に適用できます。バージョン 4.9 以降は、Metering Operator は直接置き換えなしには利用できなくなりました。

Eclipse Vert.x は、以下のメータリングラベルを使用する必要があります。

- **com.company: Red_Hat**
- **rht.prod_name: Red_Hat_Runtimes**
- **rht.prod_ver: 2023-Q1**
- **rht.comp: Vert.x**
- **rht.comp_ver: 4.3.7**
- **rht.subcomp: <leave_blank>**
- **rht.subcomp_t: application**

関連情報

- OpenShift Container Platform でのメータリングの設定および使用

3.1.2.30. OpenJDK 8 および OpenJDK 11 RHEL 8 Universal Base Images (UBI8) のサポート

Eclipse Vert.x は、RHEL 8 上の Red Hat OpenJDK 8 および Red Hat OpenJDK 11 用の OCI 準拠の ユニバーサルベースイメージ を使用して、Eclipse Vert.x アプリケーションをビルドして OpenShift にデプロイするためのサポートを導入します。

RHEL 8 OpenJDK Universal Base イメージは、RHEL 8 OpenJDK ビルダーイメージを置き換えます。RHEL 8 OpenJDK ベースイメージは、Eclipse Vert.x との使用がサポートされなくなりました。

3.2. 非推奨の機能

本セクションでは、本リリースで非推奨または削除された機能を紹介します。

3.2.1. 4.3 リリースで非推奨になる機能

次の機能は、4.3 リリースで非推奨になりました。

- Eclipse Vert.x Core

削除された要素	置き換える要素
io.vertx.core.shareddata.impl.ClusterSerializable	**io.vertx.core.shareddata.ClusterSerializable**

- Eclipse Vert.x Micrometer Metrics

非推奨となったメソッド	置き換えメソッド
io.vertx.micrometer.MicrometerMetricsOptions.getRequestsTagsProvider()	**io.vertx.micrometer.MicrometerMetricsOptions.getServerRequestsTagsProvider()**
io.vertx.micrometer.MicrometerMetricsOptions.setRequestsTagsProvider()	**io.vertx.micrometer.MicrometerMetricsOptions.setServerRequestsTagsProvider()**
io.vertx.micrometer.VertxInfluxDbOptions.getNumThreads()	置き換えるメソッドなし
io.vertx.micrometer.VertxInfluxDbOptions.setNumThreads()	置き換えるメソッドなし

- Eclipse Vert.x Web

非推奨となったメソッド	置き換えメソッド
Router.mountSubRouter(String, Router)	**Router.route(String).subRouter(Router)**
RoutingContext.getBodyAsString()	**RoutingContext.body().asString()**
RoutingContext.getBodyAsString(String encoding)	**RoutingContext.body().asString(String encoding)**
RoutingContext.getBodyAsJson()	**RoutingContext.body().asJsonObject()**
RoutingContext.getBodyAsJsonArray()	**RoutingContext.body().asJsonArray()**
RoutingContext.getBodyAsJson(int maxLength)	**RoutingContext.body().asJsonObject(int maxLength)**
RoutingContext.getBodyAsJsonArray(int maxLength)	**RoutingContext.body().asJsonArray(int maxLength)**
RoutingContext.getBody()	**RoutingContext.body().buffer()**
RouterBuilder.bodyHandler()	**RouterBuilder.rootHandler()**

- **SchemaBuilder**

削除されたメソッド	置き換えメソッド
build()	Eclipse Vert.x Json スキーマが提供する JSON 表現を使用します。以下に例を示します。 **JsonSchema schema = JsonSchema.of(dsl.toJson());**

3.2.2. 以前の 4.x リリースで非推奨になった機能

以下の機能は、以前の 4.x リリースで非推奨となったか、または削除されました。

- **HttpServerOptions**

削除されたメソッド	置き換えメソッド
getMaxWebsocketFrameSize()	getMaxWebSocketFrameSize()
setMaxWebsocketFrameSize()	setMaxWebSocketFrameSize()
getMaxWebsocketMessageSize()	getMaxWebSocketMessageSize()
setMaxWebsocketMessageSize()	setMaxWebSocketMessageSize()
getPerFrameWebsocketCompressionSupported()	getPerFrameWebSocketCompressionSupported()
setPerFrameWebsocketCompressionSupported()	setPerFrameWebSocketCompressionSupported()
getPerMessageWebsocketCompressionSupported()	getPerMessageWebSocketCompressionSupported()
setPerMessageWebsocketCompressionSupported()	setPerMessageWebSocketCompressionSupported()
getWebsocketAllowServerNoContext()	getWebSocketAllowServerNoContext()
setWebsocketAllowServerNoContext()	setWebSocketAllowServerNoContext()
getWebsocketCompressionLevel()	getWebSocketCompressionLevel()
setWebsocketCompressionLevel()	setWebSocketCompressionLevel()
getWebsocketPreferredClientNoContext()	getWebSocketPreferredClientNoContext()
setWebsocketPreferredClientNoContext()	setWebSocketPreferredClientNoContext()
getWebsocketSubProtocols()	getWebSocketSubProtocols()
setWebsocketSubProtocols()	setWebSocketSubProtocols()

- Eclipse Vert.x Web

削除された要素	置き換える要素
io.vertx.ext.web.Cookie	io.vertx.core.http.Cookie
io.vertx.ext.web.handler.CookieHandler	io.vertx.core.http.Cookie
io.vertx.ext.web.Locale	io.vertx.ext.web.LanguageHeader
RoutingContext.acceptableLocales()	RoutingContext.acceptableLanguages()
StaticHandler.create(String, ClassLoader)	---
SessionHandler.setAuthProvider(AuthProvider)	SessionHandler.addAuthProvider()
HandlebarsTemplateEngine.getHandlebars()HandlebarsTemplateEngine.getResolvers()HandlebarsTemplateEngine.setResolvers()JadeTemplateEngine.getJadeConfiguration()ThymeleafTemplateEngine.getThymeleafTemplateEngine()ThymeleafTemplateEngine.setMode()	TemplateEngine.unwrap()

- Messaging

削除されたメソッド	置き換えメソッド
MessageProducer<T>.send(T)	MessageProducer<T>.write(T)
MessageProducer.send(T,Handler)	EventBus.request(String,Object,Handler)

- EventBus

削除されたメソッド	置き換えメソッド
EventBus.send(…, Handler<AsyncResult<Message<T>>>)Message.reply(…, Handler<AsyncResult<Message<T>>>)	replyAndRequest

- Handlers

削除されたメソッド	置き換えメソッド
Future<T>.setHandler()	Future<T>.onComplete()Future<T>.onSuccess()Future<T>.onFailure()

削除されたメソッド	置き換えメソッド
HttpClientRequest.connectionHandler()	HttpClient.connectionHandler()

- JSON

削除されたフィールド/メソッド	新しいメソッド
Json.mapper() フィールド	DatabindCodec.mapper()
Json.prettyMapper() フィールド	DatabindCodec.prettyMapper()
Json.decodeValue(Buffer, TypeReference<T>)	JacksonCodec.decodeValue(Buffer, TypeReference)
Json.decodeValue(String, TypeReference<T>)	JacksonCodec.decodeValue(String, TypeReference)

- JUnit5

非推奨となったメソッド	新しいメソッド
VertxTestContext.succeeding()	VertxTestContext.succeedingThenComplete()
VertxTestContext.failing()	VertxTestContext.failingThenComplete()

- リアクティブエクステンション (Rx)

非推奨となったメソッド	新しいメソッド
WriteStreamSubscriber.onComplete()	WriteStreamSubscriber.onWriteStreamEnd()WriteStreamSubscriber.onWriteStreamError()

- サーキットブレーカー

削除されたメソッド	置き換えメソッド
CircuitBreaker.executeCommand()	CircuitBreaker.execute()
CircuitBreaker.executeCommandWithFallback()	CircuitBreaker.executeWithFallback()

- MQTT

削除されたメソッド	置き換えメソッド
MqttWill.willMessage()	**MqttWill.getWillMessage()**
MqttWill.willTopic()	**MqttWill.getWillTopic()**
MqttWill.willQos()	**MqttWill.getWillQos()**
MqttAuth.username()	**MqttAuth.getUsername()**
MqttAuth.password()	**MqttAuth.getPassword()**
MqttClientOptions.setKeepAliveTimeSeconds()	**MqttClientOptions.setKeepAliveInterval()**

- AMQP クライアント

削除されたメソッド	置き換えメソッド
AmqpClient.createReceiver(String address, Handler<AmqpMessage> messageHandler, …)	**AmqpClient createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)**
AmqpConnection createReceiver(…, Handler<AsyncResult<AmqpReceiver>> completionHandler)	**AmqpConnection createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)**
AmqpConnection createReceiver(.., Handler<AmqpMessage> messageHandler, Handler<AsyncResult<AmqpReceiver>> completionHandler)	**AmqpConnection createReceiver(String address, Handler<AsyncResult<AmqpReceiver>> completionHandler)**

- **認証および認可**

削除された要素	置き換える要素
OAuth2Options.isUseBasicAuthorizationHeader()	置き換えるメソッドなし
OAuth2Options.setUseBasicAuthorizationHeader()	置き換えるメソッドなし
OAuth2Options.getClientSecretParameterName()	置き換えるメソッドなし

削除された要素	置き換える要素
OAuth2Options.setClientSecretParameterName()	置き換えるメソッドなし
OAuth2Auth.createKeycloak()	**KeycloakAuth.create(vertx, JsonObject)()**
OAuth2Auth.create(Vertx, OAuth2FlowType, OAuth2ClientOptions)()	**OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))**
OAuth2Auth.create(Vertx, OAuth2FlowType)	**OAuth2Auth.create(vertx, new OAuth2ClientOptions().setFlow(YOUR_DESIRED_FLOW))**
User.isAuthorised()	**User.isAuthorized()**
AccessToken.refreshToken()	**AccessToken.opaqueRefreshToken()**
io.vertx.ext.auth.jwt.JWTOptions データオブジェクト	**io.vertx.ext.jwt.JWTOptions** データオブジェクト
SecretOptions クラス	**PubSecKeyOptions** クラス

非推奨となったメソッド	置き換えメソッド
OAuth2Auth.decodeToken()	**AuthProvider.authenticate()**
OAuth2Auth.introspectToken()	**AuthProvider.authenticate()**
OAuth2Auth.getFlowType()	置き換えるメソッドなし
OAuth2Auth.loadJWK()	**OAuth2Auth.jwkSet()**
Oauth2ClientOptions.isUseAuthorizationHeader()	置き換えるメソッドなし

非推奨のクラス	置き換えクラス
AbstractUser	`User.create(JsonObject)` メソッドを使用してユーザーオブジェクトを作成します。
AuthOptions	置き換えクラスなし

非推奨のクラス	置き換えクラス
JDBCAuthOptions	認証用 **JDBCAuthenticationOptions**、および承認用 **JDBCAuthorizationOptions**
JDBCHashStrategy	置き換えクラスなし
OAuth2RBAC	**AuthorizationProvider**
Oauth2Response	**WebClient** クラスの使用が推奨
KeycloakHelper	置き換えクラスなし

- **サービス検出**

削除されたメソッド	置き換えメソッド
ServiceDiscovery.create(…, Handler<ServiceDiscovery> completionHandler)	**ServiceDiscovery.create(Vertx)**
ServiceDiscovery.create(…, Handler<ServiceDiscovery> completionHandler)	**ServiceDiscovery.create(Vertx, ServiceDiscoveryOptions)**

- Eclipse Vert.x 設定

削除されたメソッド	置き換えメソッド
ConfigRetriever.getConfigAsFuture()	**retriever.getConfig()**

- MongoDB クライアント

削除されたメソッド	置き換えメソッド
MongoClient.update()	**MongoClient.updateCollection()**
MongoClient.updateWithOptions()	**MongoClient.updateCollectionWithOptions()**
MongoClient.replace()	**MongoClient.replaceDocuments()**
MongoClient.replaceWithOptions()	**MongoClient.replaceDocumentsWithOptions()**
MongoClient.remove()	**MongoClient.removeDocuments()**

削除されたメソッド	置き換えメソッド
MongoClient.removeWithOptions()	MongoClient.removeDocumentsWithOptions()
MongoClient.removeOne()	MongoClient.removeDocument()
MongoClient.removeOneWithOptions	MongoClient.removeDocumentsWithOptions()

- 共有データソースのないクライアント

非推奨となったメソッド	新しいメソッド
MongoClient.createNonShared()	MongoClient.create()
JDBCClient.createNonShared()	wJDBCClient.create()
CassandraClient.createNonShared()	CassandraClient.create()
MailClient.createNonShared()	MailClient.create()

- Hook メソッド

削除されたメソッド	新しいメソッド
Context.addCloseHook()	置き換えるメソッドなし
Context.removeCloseHook()	置き換えるメソッドなし

- クローンメソッド

削除されたメソッド	新しいメソッド
KeyCertOptions.clone()	KeyCertOptions.copy()
TrustOptions.clone()	TrustOptions.copy()
SSLEngineOptions.clone()	SSLEngineOptions.copy()

- VertxOptions

削除されたメソッド	新しいメソッド
VertxOptions.equals()	置き換えるメソッドなし

削除されたメソッド	新しいメソッド
VertxOptions.hashCode()	置き換えるメソッドなし
VertxOptions.fileResolverCachingEnabled()	**FileSystemOptions.isFileCachingEnabled()**

- **プールされたバッファー**

削除されたメソッド	新しいメソッド
TCPSSLOptions.isUsePooledBuffers()	置き換えるメソッドなし
TCPSSLOptions.setUsePooledBuffers()	置き換えるメソッドなし

第4章 リリースコンポーネント

4.1. 本リリースで導入されたサポート対象のアーティファクト

本リリースでは、テクノロジープレビューからフルサポートになったアーティファクトはありません。

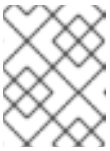
4.2. 本リリースで導入されたテクニカルプレビューアーティファクト

このセクションでは、本リリースで導入されたテクノロジープレビューアーティファクトを説明します。

4.2.1. 4.3 リリースで導入されたテクノロジープレビューアーティファクト

4.3 リリースでは、以下のアーティファクトがテクノロジープレビューとして提供されます。

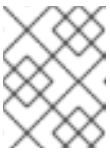
- **vertx-grpc-client**
  Eclipse Vert.x gRPC クライアントは、Eclipse Vert.x HTTP クライアントに依存する新しい Google リモートプロシージャコール (gRPC) クライアントです。Eclipse Vert.x gRPC クライアントは、サーバーと対話するための 2 つの代替方法を提供します。
  - 生成されたスタブを必要としない gRPC 要求および応答指向の API
  - gRPC チャネルを使用して生成されたスタブ

  注記

  Eclipse Vert.x gRPC クライアントは、統合された Netty ベースの gRPC クライアントに取って代わります。

- **vertx-grpc-server**
  Eclipse Vert.x gRPC サーバーは、Eclipse Vert.x HTTP サーバーに依存する新しい Google リモートプロシージャコール (gRPC) サーバーです。Eclipse Vert.x gRPC サーバーは、クライアントと対話するための 2 つの代替方法を提供します。
  - 生成されたスタブを必要としない gRPC 要求および応答指向の API
  - サービスブリッジを使用して生成されたスタブ

  注記

  Eclipse Vert.x gRPC サーバーは、統合された Netty ベースの gRPC サーバーに取って代わります。

- **vertx-grpc-common**
  Eclipse Vert.x gRPC 共通アーティファクトは、Eclipse Vert.x gRPC クライアントと Eclipse Vert.x gRPC サーバーの両方が使用する共通機能を提供します。

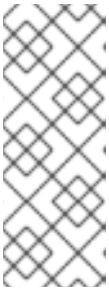
- **vertx-grpc-aggregator**
  Eclipse Vert.x gRPC Aggregator は、Project Object Model (POM) ファイルで設定されています。Eclipse Vert.x gRPC Aggregator は追加機能を提供しません。

4.2.2. 以前の 4.x リリースで導入されたテクノロジープレビューアーティファクト

以前の 4.x リリースからテクノロジープレビューとして利用できた以下のアーティファクトは、本リリースでは引き続きテクノロジープレビューとして利用できます。

- **vertx-auth-otp**
  Eclipse Vert.x OTP Auth プロバイダーは、認証を実行するためにワンタイムパスワードを使用する **AuthenticationProvider** インターフェイスの実装です。Eclipse Vert.x OTP Auth プロバイダーは Google Authenticator をサポートします。便利なライブラリーを使用して、鍵でクイックレスポンス (QR) を作成できます。base32 形式で鍵を転送することもできます。

- **vertx-oracle-client**
  Eclipse Vert.x リアクティブ Oracle クライアントは、Oracle サーバーのクライアントです。これは、データベースのスケーラビリティーに役立ち、オーバーヘッドが低い API です。API はリアクティブおよび非ブロッキングであるため、単一のスレッドで複数のデータベース接続を処理できます。

  注記

  Eclipse Vert.x リアクティブ Oracle クライアントでは、Oracle JDBC ドライバーを使用する必要があります。Red Hat では、Oracle JDBC ドライバーをサポートしていません。

  Eclipse Vert.x リアクティブ Oracle クライアントでは、JDK 11 または JDK 17 を使用する必要があります。

- **vertx-http-proxy**
  Eclipse Vert.x HTTP プロキシーはリバースプロキシーです。このモジュールを使用すると、プロキシーを簡単に作成できます。プロキシーサーバーは、オリジンサーバーからの DNS クエリーを動的に解決することもできます。

- **vertx-web-proxy**
  Eclipse Vert.x Web プロキシーを使用すると、Eclipse Vert.x Web ルーターに Eclipse Vert.x HTTP プロキシーをマウントできます。

- **vertx-opentelemetry**
  Open Telemetry のトレースがサポートされます。HTTP およびイベントバストレースに Open Telemetry を使用できます。

4.3. 本リリースで削除されたアーティファクト

本リリースでは削除になったアーティファクトはありません。

4.4. 本リリースで非推奨となったアーティファクト

本リリースでは、非推奨となったアーティファクトはありません。

第5章 修正された問題

この Eclipse Vert.x リリースには、バージョン 4.3.7 のコミュニティーリリースのすべてのバグ修正が含まれます。コミュニティーリリースで解決された問題は、Eclipse Vert.x 4.3.7 wiki ページに記載されています。

5.1. 4.3 リリースの既知の問題

このセクションでは、Eclipse Vert.x 4.3 で修正された問題について説明します。

5.1.1. クライアントとサーバーの通信に gRPC を使用すると、スレッドが io.vertx.coreVertxException でブロックされる

Eclipse Vert.x 4.3.4 以前では、クライアントとサーバー間の Google リモートプロシージャコール (gRPC) 通信がスレッドをブロックし、**io.vertx.core.VertxException** エラーが発生していました。この問題は、使用可能なイベントループスレッドの数が不十分で、Eclipse Vert.x gRPC サーバー (**vertx-grpc-server**) または Eclipse Vert.x gRPC クライアント (**vertx-grpc-client**) がセルフデッドロックした場合に発生しました。

この問題は Eclipse Vert.x 4.3.5 リリースで解決されました。SSL 初期化の内部コンテキストが、イベントループコンテキストではなくワーカーコンテキストを使用するようになりました。

5.1.2. ROWID でテーブルデータを検索する場合の JDBCClient エラー

Eclipse Vert.x 4.2 では、**ROWID** を使用してテーブルデータを取得しようとすると、Oracle JDBC ドライバーを持つ **vertx-jdbc-client** が **java.sql.SQLException** を出力します。この問題は、Eclipse Vert.x 4.2 で、**ROWID** がバイトの配列として使用可能であったために発生しました。Eclipse Vert.x の以前のリリースでは、**ROWID** は文字列タイプとして使用可能でした。

この問題は Eclipse Vert.x 4.3.1 リリースで解決されました。Eclipse Vert.x 4.2 は、プレーンな Java 型に基づくカスタム型キャスティングを行いました。通常、この動作では正しい結果が得られますが、特殊なデータベースタイプを誤って識別する可能性があります。カスタムタイプキャストは最新の JDBC ドライバーの組み込み機能ではないため、JDBC クライアントはドライバーに依存して、古い Eclipse Vert.x ヒューリスティックよりも適したベンダー固有のキャストを実行するようになりました。

5.1.3. ROWID 解析時の JDBCPool エラー

Eclipse Vert.x 4.2 では、**ROWID** を解析しようとすると、JDBC プールは **java.lang.UnsupportedOperationException** が出力されました。この問題は、Eclipse Vert.x 4.2 では、**ROWID** をバイトの配列として直接解析できなかったために発生しました。{VertX) の以前のリリースでは、**ROWID** を文字列型として解析できました。

この問題は、前のセクションで説明したものと同じソリューションに基づいて、Eclipse Vert.x 4.3.1 リリースで解決されています。

5.1.4. PostgreSQL JDBC ドライバー 9.0 以降を使用すると、ストアドプロシージャコールで予期しない結果が発生する

Eclipse Vert.x 4.2 では、PostgreSQL JDBC ドライバー 9.0 以降を使用する **vertx-jdbc-client** は、ストアドプロシージャの呼び出しで予期しない結果を生成します。この問題は、Eclipse Vert.x 4.2 で、**vertx-jdbc-client** が、最新の PostgreSQL データベースドライバーとサーバーが呼び出し可能なステートメントを実行するときに必要とする明示的な SQL 型情報をサポートしていなかったために発生しました。

この問題は Eclipse Vert.x 4.3.1 リリースで解決されました。呼び出し可能な **ResultSet** メタデータは、クエリーに関係するすべてのソース、最初のデータベース応答、および応答の一部を形成する個々の結果セットから抽出されるようになりました。完全な情報により、JDBC クライアントは列内のデータの型を正しく識別し、正しいキャストを実行できます。

5.2. 以前の 4.X リリースで修正された問題

このセクションでは、以前の Eclipse Vert.x 4.x リリースで修正された問題について説明します。

5.2.1. GraphQL ビルドに含まれる Google Guava クラス

Eclipse Vert.x 4.0.0 および 4.0.2 リリースでは、**vertx-web-graphql** 依存関係を使用できませんでした。これは、バージョン 16.1.0.redhat-00001 を持つ GraphQL Java の不完全なビルドが使用されたためです。不完全な GraphQL ビルドには、Guava クラスがありませんでした。

この問題は Eclipse Vert.x 4.0.3 リリースで解決されました。このリリースには、Guava クラスの完全なビルドである GraphQL Java 16.1.0.redhat-00002 バージョンが含まれます。これらの Guava クラスは shaded jar になります。

5.2.2. Eclipse Vert.x ビルドで利用可能な vertx-opentracing

vertx-opentracing 依存関係は、Eclipse Vert.x 4.0.0 でテクノロジープレビュー機能として導入されました。ただし、この依存関係は Eclipse Vert.x 4.0.0 および 4.0.2 リリースでは利用できませんでした。

この問題は Eclipse Vert.x 4.0.3 リリースで解決されました。このリリースには **vertx-opentracing** 依存関係が含まれます。

第6章 既知の問題

6.1. 4.3 リリースの既知の問題

このリリースに影響する既知の問題はありません。

6.2. 以前の 4.X リリースでの既知の問題

このセクションでは、以前の Eclipse Vert.x 4.x リリースからの既知の問題について説明します。

6.2.1. RH-SSO が発行するトークンは、Eclipse Vert.x 4.2 に移行した後、OAuth2 検証に失敗します。

説明

Red Hat Single Sign-On (RH-SSO) を認証プロバイダーとして使用する場合、Eclipse Vert.x の以前のリリースで有効であったトークンは、Eclipse Vert.x 4.2 に移行すると検証チェックに失敗します。

原因

Eclipse Vert.x 4.2 では、OAuth2 認証は、Eclipse Vert.x の以前のリリースよりも厳密なセキュリティー検証を提供します。

回避策

デフォルトでは、RH-SSO はクライアント ID ではなく **account** オーディエンスでトークンを発行します。RH-SSO を認証プロバイダーとして使用している場合、Eclipse Vert.x 4.2 に移行した後、**account** オーディエンスとクライアント ID を明示的に JWTOptions 設定に追加し、トークンが正常に検証されるようにする必要があります。

6.2.2. JDK 8 で **vertx-oracle-client** を使用するとコンパイルエラーが発生する

説明

Eclipse Vert.x 4.2 では、OpenJDK 8 を使用すると、**vertx-oracle-client** が **bad class file** コンパイルエラーを出力します。

原因

Eclipse Vert.x 4.2 では、**vertx-oracle-client** は OpenJDK 11 以降で動作するように設計されています。

回避策

OpenJDK 11 または OpenJDK 17 を使用します。

6.2.3. **KubernetesServiceImporter()** を Eclipse Vert.x Reactive Extensions (Rx) に直接登録できない

説明

Eclipse Vert.x の Reactive Extensions (Rx) で **KubernetesServiceImporter()** を直接登録することはできません。

原因

サービスインポーターには生成された RxJava 2 実装がありません。

回避策

以下の例のように、**KubernetesServiceImporter** のインスタンスを作成し、**{@link io.vertx.reactivex.servicediscovery.spi.ServiceImporter}** でカプセル化する必要があります。

```
{@link
examples.RxServiceDiscoveryExamples#register(io.vertx.reactivex.servicediscovery.ServiceDiscovery)}
```

以下の例は、Eclipse Vert.x Reactive Extensions (Rx) で **KubernetesServiceImporter()** を登録する方法を示しています。

```
ServiceDiscovery discovery = ServiceDiscovery.create(vertx);
discovery.getDelegate().registerServiceImporter(new KubernetesServiceImporter(), new
JsonObject());
```

6.2.4. IBM Z および IBM Power Systems では、Red Hat AMQ Streams イメージが利用できない

Red Hat AMQ Streams Operator および Kafka イメージは、IBM Z および IBM Power Systems では利用できません。イメージは利用できないため、**vertx-kafka-client** モジュールは IBM Z および IBM Power Systems 上の AMQ Streams と動作することが認定されていません。

6.2.5. TLS プロトコルのバージョンが一致しないため、RHEL 8 ベースのデータベースアプリケーションと RHEL 7 ベースの MySQL 5.7 データベース間の接続が失敗する

説明

RHEL 8 ベースの OpenJDK ビルダーイメージで構築されたアプリケーションコンテナーと、RHEL 7 ベースの MySQL 5.7 コンテナーイメージ上に構築されたデータベースコンテナーとの間で、OpenSSL を使用して TLS でセキュア化された接続を開くと、ランタイム時に **javax.net.ssl.SSLHandshakeException** によって接続に失敗します。詳細は、JIRA の問題 を参照してください。

```
...
Caused by: javax.net.ssl.SSLHandshakeException: No appropriate protocol (protocol is disabled or
cipher suites are inappropriate)
...
```

原因

この問題は、RHEL 7 から RHEL 8 でサポートされる最新の TLS プロトコルバージョンが異なるために発生します。RHEL 7 の TLS 実装は、TLS プロトコルバージョン 1.0 (非推奨)、1.1、および 1.2 に対応しています。RHEL 8 の TLS 実装は、TLS プロトコルバージョン 1.3 にも対応しています。これは、RHEL 8 ベースのビルダーイメージで使用されるデフォルトの TLS バージョンでもあります。この不一致により、TLS ハンドシェイクのネゴシエーション中に、アプリケーションコンポーネント間で TLS プロトコルバージョンの不一致が発生し、アプリケーションとデータベースコンテナーとの間の接続が失敗する可能性があります。

回避策

上記の問題を防ぐには、データベース接続文字列の両方のオペレーティングシステムバージョンでサポートされる TLS プロトコルバージョンを手動で指定します。以下に例を示します。

```
jdbc:mysql://testdb-mysql:3306/testdb?enabledTLSProtocols=TLSv1.2
```

6.2.6. False Connection がアプリケーションエンドポイントを呼び出す際にピアエラーメッセージによってリセットされる

curl ツールまたは Java HTTP クライアントのいずれかを使用して Eclipse Vert.x アプリケーションのエンドポイントで HTTP リクエストを行うと、リクエストごとに以下のエラーがログに出力されます。

```
io.vertx.core.net.impl.ConnectionBase
SEVERE: java.io.IOException: Connection reset by peer
```

この動作は、Netty アプリケーションフレームワークと、OpenShift によって使用される HAProxy ロードバランサーの対話によって生じます。このエラーは、HAProxy が閉じずに既存の HTTP 接続が再使用されるため発生します。エラーメッセージがログに記録されても、エラー状態は発生しません。HTTP リクエストが正しく処理され、アプリケーションは予想通りに応答します。

第7章 本リリースに関連するアドバイザリー

本リリースに含まれる拡張機能、バグ修正、および CVE 修正を文書化するために、以下のアドバイザリーが発行されています。

- RHSA-2023:0577