Red Hat build of Eclipse Vert.x 4.0

Eclipse Vert.x 4.0 移行ガイド

ガイド

Last Updated: 2023-06-04

Red Hat build of Eclipse Vert.x 4.0 Eclipse Vert.x 4.0 移行ガイド

ガイド

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

本ガイドでは、Eclipse Vert.x 3.x アプリケーションを Eclipse Vert.x 4 にアップグレードする方法を説明します。

目次

はじめに

本ガイドでは、Eclipse Vert.x 4 リリースの更新を説明します。この情報を使用して Eclipse Vert.x 3.x アプリケーションを Eclipse Vert.x 4 にアップグレードします。本リリースにおける新機能、非推奨となった新機能、およびサポート対象外になった機能に関する情報を提供します。

アプリケーションで使用されるモジュールに応じて、関連セクションを読み取り、Eclipse Vert.x 4 の変更の詳細を確認することができます。

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、**Multi-page 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の近くに表示される **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 ECLIPSE VERT.X を使用するようにアプリケーションを設定

Eclipse Vert.x を使用するようにアプリケーションを設定する場合は、アプリケーションのルートディレクトリーにある **pom.xml** ファイルの Eclipse Vert.x BOM (Bill of Materials) アーティファクトを参照する必要があります。BOM は、アーティファクトの正しいバージョンを設定するのに使用されます。

前提条件

- Maven ベースのアプリケーション

手順

1. **pom.xml** ファイルを開き、**io.vertx:vertx-dependencies** アーティファクトを **<dependencyManagement>** セクションに追加します。**type** を **pom** として指定し、**scope** を **import** として指定します。

```
<project>
 ...
 <dependencyManagement>
  <dependencies>
   <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-dependencies</artifactId>
    <version>${vertx.version}</version>
    <type>pom</type>
    <scope>import</scope>
   </dependency>
  </dependencies>
 </dependencyManagement>
 ...
</project>
```

2. 以下のプロパティーを追加して、使用する Eclipse Vert.x および Eclipse Vert.x Maven Plugin のバージョンを追跡します。
プロパティーを使用して、リリースごとに変更する値を設定できます。たとえば、製品またはプラグインのバージョンです。

```
<project>
 ...
 <properties>
  <vertx.version>${vertx.version}</vertx.version>
  <vertx-maven-plugin.version>${vertx-maven-plugin.version}</vertx-maven-plugin.version>
 </properties>
 ...
</project>
```

3. アプリケーションのパッケージ化に使用されるプラグインとして **vertx-maven-plugin** を指定します。

```
<project>
 ...
 <build>
  <plugins>
   ...
```

```
      <plugin>
        <groupId>io.reactiverse</groupId>
        <artifactId>vertx-maven-plugin</artifactId>
        <version>${vertx-maven-plugin.version}</version>
        <executions>
          <execution>
            <id>vmp</id>
            <goals>
              <goal>initialize</goal>
              <goal>package</goal>
            </goals>
          </execution>
        </executions>
        <configuration>
          <redeploy>true</redeploy>
        </configuration>
      </plugin>
      ...
    </plugins>
  </build>
  ...
</project>
```

4. **repositories** および **pluginRepositories** を追加して、アプリケーションをビルドするためのアーティファクトおよびプラグインが含まれるリポジトリーを指定します。

```
<project>
...
  <repositories>
    <repository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </repository>
  </repositories>
  <pluginRepositories>
    <pluginRepository>
      <id>redhat-ga</id>
      <name>Red Hat GA Repository</name>
      <url>https://maven.repository.redhat.com/ga/</url>
    </pluginRepository>
  </pluginRepositories>
...
</project>
```

関連情報

- Eclipse Vert.x アプリケーションのパッケージ化に関する詳細は、Vert.x Maven プラグイン のドキュメントを参照してください。

第2章 ECLIPSE VERT.X について

Eclipse Vert.x は、Java 仮想マシンで実行する、リアクティブで、ブロックされない、非同期アプリケーションを作成するのに使用されるツールキットです。(JVM)。これには、リアクティブアプリケーションの作成に役立つ複数のコンポーネントが含まれています。これは、cloud-native として設計されています。

Eclipse Vert.x は非同期アプリケーションをサポートするため、大量のメッセージ、大規模なイベント処理、HTTP の対話などがあるアプリケーションを作成するために使用できます。

第3章 ECLIPSE VERT.X 4 で変更された内容

本セクションでは、Eclipse Vert.x 4 リリースと 3.x リリースの基本的な相違点を説明します。

3.1. 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。callback と future メソッドの組み合わせを使用して、callback ベースのアプリケーションを Eclipse Vert.x 4 に移行できます。ただし、非同期操作の callback の使用を継続することもできます。

以下の例は、Eclipse Vert.x 3.x リリースで callback が非同期操作に使用される方法を示しています。

```
WebClient client = WebClient.create(vertx);
HttpRequest request = client.get("/resource");

request.send(ar -> {
  if (ar.succeeded()) {
    HttpResponse response = ar.result();
  } else {
    Throwable error = ar.cause();
  }
});
```

以下の例は、Eclipse Vert.x 4 で callback メソッドと future メソッドを非同期操作に使用する方法を示しています。

```
WebClient client = WebClient.create(vertx);
HttpRequest request = client.get("/resource");

Future<HttpResponse> response = request.send();

response.onComplete(ar -> {
  if (ar.succeeded()) {
    HttpResponse response = ar.result();
  } else {
    Throwable failure = ar.cause();
  }
});
```

future でエラー処理が改善されました。callback では構成の各段階で障害を処理する必要がありますが、future は障害を一度処理できます。基本的なアプリケーションでは、callback と future の使用で明確な違いに気づかない場合があります。

以下の例は、callback を使用して 2 つの非同期操作を作成する方法を示しています。すべての構成でエラーが処理されることが確認できます。

```
client.get("/resource1").send(ar1 -> {
  if (ar1.succeeded()) {
    HttpResponse response = ar.result();
    JsonObject json = response.body();
    client.put("/resource2").sendJsonObject(ar2 -> {
      if (ar2.succeeded()) {
        // Handle final result
```

```
      } else {
        Throwable failure2 = ar.cause();
      }
    });
  } else {
    Throwable failure1 = ar.cause();
  }
});
```

以下の例は、Eclipse Vert.x 4 で callback および future を使用して 2 つの非同期操作を作成する方法を示しています。エラーは最後に一度のみ処理されます。

```
Future<HttpResponse> fut1 = client.get("/resource1").send();

Future<HttpResponse> fut2 = fut1.compose(response ->
client.put("/resource2").sendJsonObject(response.body()));

fut2.onComplete(ar -> {
  if (ar.succeeded()) {
    // Handle final result
  } else {
    Throwable failure = ar.cause();
  }
});
```

3.2. JACKSON DATABIND ライブラリーの依存関係がない

Eclipse Vert.x の JSON 機能は、Jackson ライブラリーに依存します。Jackson Databind ライブラリーは JSON のオブジェクトマッピングを有効にします。

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。

- オブジェクトマッピング JSON の場合は、**com.fasterxml.jackson.core:jackson-databind** jar のプロジェクト記述子に依存関係を明示的に追加する必要があります。

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

  JSON のオブジェクトマッピングを使用しない場合は、この依存関係を削除できます。

- オブジェクトマッピング JSON を使用しない場合、Jackson Databind ライブラリーは必要ありません。この jar を使用せずにアプリケーションを実行できます。

3.3. 非推奨と削除の処理

一部の機能および関数は 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。

- 一部の API は Eclipse Vert.x 3.x リリースで非推奨となり、そのリリースで同等の API が新たに提供されました。
- 非推奨の API は Eclipse Vert.x 4 で削除されました。

アプリケーションが非推奨の API を使用する場合は、新しい API を使用するようにアプリケーションを更新する必要があります。これは、アプリケーションを製品の最新バージョンに移行するのに役立ちます。

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。

以下の例は、Eclipse Vert.x 3.x リリースで非推奨となった EventBus メソッドを示しています。

```
// Send was deprecated in Vert.x 3.x release
vertx.eventBus().send("some-address", "hello world", ar -> {
  // Handle response here
});
```

Eclipse Vert.x 4 で **send(String,String,Handler<AsyncResult<Message>>)** メソッドが、**request(String,String,Handler<AsyncResult<Message>>**) に置き換えられています。

以下の例は、新しいメソッドを使用するようアプリケーションを更新する方法を示しています。

```
// New method can be used in Vert.x 3.x and Vert.x 4.x releases
vertx.eventBus().request("some-address", "hello world", ar -> {
  // Handle response here
});
```

第4章 一般的なコンポーネントの変更

本セクションでは、基本的な Eclipse Vert.x コンポーネントの変更点を説明します。

4.1. メッセージングの変更点

本セクションでは、メッセージングメソッドの変更点を説明します。

4.1.1. 書き込みストリームの書き込みメソッドおよび終了メソッドが fluent でなくなる

WriteStream<T>.write() メソッドおよび **WriteStream<T>.end()** メソッドは fluent ではなくなりました。

- 書き込みおよび終了 callback メソッドは **void** を返します。
- その他の書き込みメソッドおよび終了メソッドは **Future<Void>** を返します。

これは、重大な変更です。書き込みストリームに fluent 側面を使用している場合は、アプリケーションを更新します。

4.1.2. MessageProducer が WriteStream を拡張しない

MessageProducer インターフェースが **WriteStream** インターフェースを拡張しません。

以前のリリースの Eclipse Vert.x では、**MessageProducer** インターフェースは **WriteStream** インターフェースを拡張しました。**MessageProducer** インターフェースは、メッセージのバックプレッシャーに対する制限されたサポートを提供していました。クレジットリークにより、メッセージプロデューサーのクレジットが減ります。これらのリークがすべてクレジットを使用していると、メッセージは送信されません。

ただし、**MessageConsumer** は、**ReadStream** の拡張機能を継続します。**MessageConsumer** が一時停止し、保留中のメッセージキューが満杯になると、メッセージは破棄されます。これにより、Rx ジェネレーターとのインテグレーションが継続され、消費するパイプラインのメッセージが作成されます。

4.1.3. MessageProducerから送信メソッドを削除

MessageProducer インターフェースの送信メソッドが削除されました。

MessageProducer<T>.send(T) の代わりに **MessageProducer<T>.write(T)** メソッドを使用し、**MessageProducer.send(T,Handler)** の代わりに **EventBus.request(String,Object,Handler)** メソッドを使用してください。

4.2. EVENTBUS の変更点

以下のセクションでは、EventBus の変更点を説明します。

4.2.1. EventBus の request-response 送信メソッドを削除

EventBus.send(..., Handler<AsyncResult<Message<T>>>) メソッドおよび **Message.reply(..., Handler<AsyncResult<Message<T>>>)** メソッドが削除されました。これらのメソッドにより、Eclipse Vert.x 4 でオーバーロードの問題が生じました。**Future<Message<T>>** メソッドのバージョンが、fire および forget バージョンと競合します。

request-response メッセージングパターンでは、新しい **request** メソッドおよび **replyAndRequest** メソッドを使用する必要があります。

- **EventBus.send(…, Handler<AsyncResult<Message<T>>>)** の代わりに **EventBus.request(…, Handler<AsyncResult<Message<T>>>)** メソッドを使用してメッセージを送信します。
- **Message.reply(…, Handler<AsyncResult<Message<T>>>)** の代わりに **Message.replyAndRequest(…, Handler<AsyncResult<Message<T>>>)** メソッドを使用して、メッセージに応答します。

以下の例は、Eclipse Vert.x 3.x リリースのメッセージへの要求および応答を示しています。

Request

```
eventBus.send("the-address", body, ar -> ...);
```

応答

```
eventBus.consumer("the-address", message -> {
      message.reply(body, ar -> ...);
});
```

以下の例は、Eclipse Vert.x 4 のメッセージへの要求および応答を示しています。

Request

```
eventBus.request("the-address", body, ar -> ...);
```

応答

```
eventBus.consumer("the-address", message -> {
  message.replyAndRequest(body, ar -> ...);
});
```

4.3. FUTURE の変更点

本セクションでは、future の変更点を説明します。

4.3.1. future の複数のハンドラーのサポート

Eclipse Vert.x 4 以降では、future で複数のハンドラーがサポートされます。1 つのハンドラーを設定するために使用される **Future<T>.setHandler()** メソッドが削除されています。代わりに、**Future<T>.onComplete()** メソッド、**Future<T>.onSuccess()** メソッド、および **Future<T>.onFailure()** メソッドを使用して、それぞれアクションの完了、成功、および失敗の結果でハンドラーを呼び出します。

以下の例は、Eclipse Vert.x 3.x リリースでハンドラーを呼び出す方法を示しています。

```
Future<String> fut = getSomeFuture();
fut.setHandler(ar -> ...);
```

以下の例は、Eclipse Vert.x 4 で新しい **Future<T>.onComplete()** メソッドを呼び出す方法を示しています。

```
Future<String> fut = getSomeFuture();
fut.onComplete(ar -> ...);
```

4.3.2. future の completer() メソッドを削除

以前のリリースの Eclipse Vert.x では、**Future.completer()** メソッドを使用して、**Future** と関連付けられていた **Handler<AsyncResult<T>>** にアクセスします。

Eclipse Vert.x 4 では、**Future<T>.completer()** メソッドが削除されました。**Future<T>** は、**Handler<AsyncResult<T>>** を直接拡張します。**Future** オブジェクトを使用してすべてのハンドラーメソッドにアクセスできます。**Future** オブジェクトはハンドラーでもあります。

4.3.3. HTTP クライアント要求の接続ハンドラーメソッドを削除

HttpClientRequest.connectionHandler() メソッドが削除されました。代わりに **HttpClient.connectionHandler()** メソッドを使用して、アプリケーションのクライアントリクエストの接続ハンドラーを呼び出します。

以下の例は、**HttpClientRequest.connectionHandler()** メソッドが Eclipse Vert.x 3.x リリースでどのように使用されたかを示しています。

```
client.request().connectionHandler(conn -> {
  // Connection related code
}).end();
```

以下の例は、Eclipse Vert.x 4 で新しい **HttpClient.connectionHandler()** メソッドを使用する方法を示しています。

```
client.connectionHandler(conn -> {
  // Connection related code
});
```

4.4. VERTICLE の変更点

本セクションでは、verticle の変更点を説明します。

4.4.1. create verticle メソッドの更新

Eclipse Vert.x の以前のリリースでは、**VerticleFactory.createVerticle()** メソッドが verticle を非同期的にインスタンス化しました。Eclipse Vert.x 4 以降では、メソッドは非同期的に verticle をインスタンス化し、単一の verticle インスタンスではなく、コールバック **Callable<Verticle>** を返します。この改善により、アプリケーションはこのメソッドを一度呼び出し、複数のインスタンスを作成するために callable を複数回呼び出すことが可能になります。

以下のコードは、Eclipse Vert.x 3.x リリースで verticle がどのようにインスタンス化されたかを示しています。

```
Verticle createVerticle(String verticleName, ClassLoader classLoader) throws Exception;
```

以下のコードは、Eclipse Vert.x 4 で verticle がインスタンス化される方法を示しています。

```
void createVerticle(String verticleName, ClassLoader classLoader, Promise<Callable<Verticle>>
promise);
```

4.4.2. factory クラスおよびメソッドの更新

VerticleFactory クラスが簡素化されました。ファクトリーはネストされたデプロイメントを使用して verticle をデプロイすることができるため、クラスは識別子の初期解決を必要としません。

既存のアプリケーションが factory を使用する場合、Eclipse Vert.x 4 では、予想される完了または失敗時に callable を使用するようにコードを更新できます。呼び出し可能なものは複数回呼び出すことができます。

以下の例は、Eclipse Vert.x 3.x アプリケーションの既存のファクトリーを示しています。

```
return new MyVerticle();
```

以下の例は、Eclipse Vert.x 4 で promise を使用するように既存の factory を更新する方法を示しています。

```
promise.complete(() -> new MyVerticle());
```

factory がコードをブロックするようにするには、**Vertx.executeBlocking()** メソッドを使用します。factory がブロックコードを受け取ると、promise を解決し、promise から verticle インスタンスを取得する必要があります。

4.4.3. マルチスレッドのワーカー verticle を削除

マルチスレッドのワーカーの verticle プロイメントオプションが削除されました。この機能は Eclipse Vert.x event-bus とのみ使用できます。HTTP などの他の Eclipse Vert.x コンポーネントはこの機能をサポートしません。

順序のない **Vertx.executeBlocking()** メソッドを使用して、マルチスレッドワーカーのデプロイメントと同じ機能を実現します。

4.5. スレッドの変更点

本セクションでは、スレッドの変更点を説明します。

4.5.1. 非 Eclipse Vert.x スレッドのコンテキストアフィニティー

Vertx.getOrCreateContext() メソッドは、非 Eclipse Vert.x スレッドごとに単一のコンテキストを作成します。非 Eclipse Vert.x スレッドは、コンテキストの初回作成時にコンテキストに関連付けられます。以前のリリースでは、Eclipse Vert.x 以外のスレッドからメソッドが呼び出されるたびに、新しいコンテキストが作成されていました。

```
new Thread(() -> {
  assertSame(vertx.getOrCreateContext(), vertx.getOrCreateContext());
}).start();
```

アプリケーションが呼び出しごとに作成される新しいコンテキストに暗黙的に依存しない限り、この変更はアプリケーションに影響を与えません。

以下の例では、各ブロックコードが異なるコンテキストで呼び出されるため、n ブロックが同時に実行されます。

```
for (int i = 0;i < n;i++) {
  vertx.executeBlocking(block, handler);
}
```

Eclipse Vert.x 4 で同じ結果を取得するには、コードを更新する必要があります。

```
for (int i = 0;i < n;i++) {
  vertx.executeBlocking(block, false, handler);
}
```

4.6. HTTP の変更点

本セクションでは、HTTP メソッドの変更点を説明します。

4.6.1. Eclipse Vert.x HTTP メソッドにおける一般的な更新点

以下のセクションでは、Eclipse Vert.x HTTP メソッドにおけるその他の更新を説明します。

4.6.1.1. WebSocket の HTTP メソッドの更新

WebSocket の変更点は次のとおりです。

- メソッド名で **WebSocket** という用語の使用に一貫性がありませんでした。メソッド名に、**WebSocket** ではなく **Websocket** などの誤った大文字が含まれていました。以下のクラスで **WebSocket** の使用に一貫性のないメソッドが削除されました。代わりに正しい大文字を持つ新しいメソッドを使用してください。
  - **HttpServerOptions** クラスの以下のメソッドが削除されました。

削除されたメソッド	新しいメソッド
getMaxWebsocketFrameSize()	**getMaxWebSocketFrameSize()**
setMaxWebsocketFrameSize()	**setMaxWebSocketFrameSize()**
getMaxWebsocketMessageSize()	**getMaxWebSocketMessageSize()**
setMaxWebsocketMessageSize()	**setMaxWebSocketMessageSize()**
getPerFrameWebsocketCompressionSupported()	**getPerFrameWebSocketCompressionSupported()**
setPerFrameWebsocketCompressionSupported()	**setPerFrameWebSocketCompressionSupported()**
getPerMessageWebsocketCompressionSupported()	**getPerMessageWebSocketCompressionSupported()**

削除されたメソッド	新しいメソッド
setPerMessageWebsocketCompressionSupported()	setPerMessageWebSocketCompressionSupported()
getWebsocketAllowServerNoContext()	getWebSocketAllowServerNoContext()
setWebsocketAllowServerNoContext()	setWebSocketAllowServerNoContext()
getWebsocketCompressionLevel()	getWebSocketCompressionLevel()
setWebsocketCompressionLevel()	setWebSocketCompressionLevel()
getWebsocketPreferredClientNoContext()	getWebSocketPreferredClientNoContext()
setWebsocketPreferredClientNoContext()	setWebSocketPreferredClientNoContext()
getWebsocketSubProtocols()	getWebSocketSubProtocols()
setWebsocketSubProtocols()	setWebSocketSubProtocols()

WebSocket サブプロトコルの新しいメソッドは、項目を保存するためにコンマ区切りの文字列の代わりに **List<String>** データ型を使用します。

- **HttpClientOptions** クラスの以下のメソッドが削除されました。

削除されたメソッド	置き換えメソッド
getTryUsePerMessageWebsocketCompression()	getTryUsePerMessageWebSocketCompression()
setTryUsePerMessageWebsocketCompression()	setTryUsePerMessageWebSocketCompression()
getTryWebsocketDeflateFrameCompression()	getTryWebSocketDeflateFrameCompression()
getWebsocketCompressionAllowClientNoContext()	getWebSocketCompressionAllowClientNoContext()
setWebsocketCompressionAllowClientNoContext()	setWebSocketCompressionAllowClientNoContext()
getWebsocketCompressionLevel()	getWebSocketCompressionLevel()
setWebsocketCompressionLevel()	setWebSocketCompressionLevel()

削除されたメソッド	置き換えメソッド
getWebsocketCompressionRequestServerNoContext()	**getWebSocketCompressionRequestServerNoContext()**
setWebsocketCompressionRequestServerNoContext()	**setWebSocketCompressionRequestServerNoContext()**

- **HttpServer** クラスの以下のハンドラーメソッドが削除されました。

非推奨となったメソッド	新しいメソッド
websocketHandler()	**webSocketHandler()**
websocketStream()	**webSocketStream()**

- **WebsocketRejectedException** は非推奨になりました。メソッドは、代わりに **UpgradeRejectedException** を出力します。
- **HttpClient webSocket()** メソッドは、**Handler** または **Handler<Throwable>** の代わりに **Handler<AsyncResult<WebSocket>>** を使用します。
- また、**WebSocketConnectOptions** クラスのメソッドを使用して HTTP クライアントを WebSocket に接続する多重定義されたメソッドの数も削減されました。
- **HttpServerRequest.upgrade()** メソッドが削除されました。このメソッドは同期的でした。代わりに、新しいメソッド **HttpServerRequest.toWebSocket()** を使用してください。この新しいメソッドは非同期的です。

  以下の例は、Eclipse Vert.x 3.x での同期メソッドの使用を示しています。

```
// 3.x
server.requestHandler(req -> {
  WebSocket ws = req.upgrade();
});
```

  以下の例は、Eclipse Vert.x 4 での非同期メソッドの使用を示しています。

```
// 4.0
server.requestHandler(req -> {
  Future<WebSocket> fut = req.toWebSocket();
  fut.onSuccess(ws -> {
  });
});
```

4.6.1.2. WebSocket 接続の数の設定

Eclipse Vert.x 3.x では、HTTP クライアントプールサイズを使用してアプリケーションで WebSocket 接続の最大数を定義することができます。値アクセサーメソッド **HttpClientOptions.maxPoolSize()** は、WebSocket 接続の取得および設定に使用されました。デフォルトの接続数は、エンドポイントごとに 4 に設定されました。

以下の例は、Eclipse Vert.x 3.x で WebSocket 接続が設定される方法を示しています。

```
// 3.x
options.setMaxPoolSize(30); // Maximum connection is set to 30 for each endpoint
```

しかし、Eclipse Vert.x 4 では、使用後に接続が閉じられるため、WebSocket TCP 接続のプールはありません。アプリケーションは、HTTP リクエストに異なるプールを使用します。値のアクセサーメソッド **HttpClientOptions.maxWebSockets()** を使用して WebSocket 接続を取得および設定します。デフォルトの接続数は、エンドポイントごとに 50 に設定されます。

以下の例は、Eclipse Vert.x 4 で WebSocket 接続を設定する方法を示しています。

```
// 4.0
options.setMaxWebSockets(30); // Maximum connection is set to 30 for each endpoint
```

4.6.1.3. HttpMethod がインターフェースとして利用可能

HttpMethod は新しいインターフェースとして利用できます。

以前のリリースの Eclipse Vert.x では、**HttpMethod** は列挙型として宣言されていました。列挙として、HTTP の拡張性が制限されます。さらに、このタイプの他の HTTP メソッドを直接提供できませんでした。サーバーおよびクライアントの HTTP 要求時に、**HttpMethod.OTHER** の値を **rawMethod** 属性と共に使用する必要がありました。

スイッチブロックで **HttpMethod** 列挙データ型を使用している場合は、以下のコードを使用してアプリケーションを Eclipse Vert.x 4 に移行できます。

以下の例は、Eclipse Vert.x 3.x リリースの switch ブロックを示しています。

```
switch (method) {
  case GET:
    ...
    break;
  case OTHER:
    String s = request.getRawMethod();
    if (s.equals("PROPFIND") {
      ...
    } else ...
}
```

以下の例は、Eclipse Vert.x 4 の switch ブロックを示しています。

```
switch (method.name()) {
  case "GET":
    ...
    break;
  case "PROPFIND";
    ...
    break;
}
```

Eclipse Vert.x 4 で以下のコードを使用することもできます。

```
HttpMethod PROPFIND = HttpMethod.valueOf("PROPFIND");
```

```
if (method == HttpMethod.GET) {
  ...
} else if (method.equals(PROPFIND)) {
  ...
} else {
  ...
}
```

アプリケーションで **HttpMethod.OTHER** 値を使用している場合は、以下のコードを使用してアプリケーションを Eclipse Vert.x 4 に移行します。

以下の例は、Eclipse Vert.x 3.x リリースのコードを示しています。

```
client.request(HttpMethod.OTHER, ...).setRawName("PROPFIND");
```

以下の例は、Eclipse Vert.x 4 のコードを示しています。

```
client.request(HttpMethod.valueOf("PROPFIND"), ...);
```

4.6.2. HTTP クライアントの変更点

本セクションでは、HTTP クライアントの変更点を説明します。

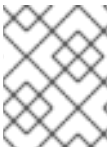
以下のタイプの Eclipse Vert.x クライアントを利用できます。

Eclipse Vert.x Web クライアント

アプリケーションが Web 指向の場合は、Eclipse Vert.x Web クライアントを使用します。たとえば、REST、HTTP ペイロードのエンコーディングおよびデコーディング、HTTP ステータス応答コードの解釈などです。

Eclipse Vert.x HTTP クライアント

アプリケーションが HTTP プロキシーとして使用される場合は、Eclipse Vert.x HTTP クライアントを使用します。たとえば、API ゲートウェイとしてです。HTTP クライアントが更新され、Eclipse Vert.x 4 で改善されました。



注記

Eclipse Vert.x Web クライアントは Eclipse Vert.x HTTP クライアントに基づいています。

4.6.2.1. アプリケーションの Eclipse Vert.x Web クライアントへの移行

Web クライアントは Eclipse Vert.x 3.4.0 リリースから入手できました。Eclipse Vert.x 4 では、Web クライアントに変更点はありません。

クライアントは、簡素化された HTTP 対話と、Eclipse Vert.x HTTP クライアントでは利用できない HTTP セッション、JSON エンコーディングおよびデコーディング、応答述語などの追加機能を提供します。

以下の例は、Eclipse Vert.x 3.x リリースで HTTP クライアントを使用する方法を示しています。

```
HttpClientRequest request = client.get(80, "example.com", "/", response -> {
  int statusCode = response.statusCode();
```

```
  response.exceptionHandler(err -> {
    // Handle connection error, for example, connection closed
  });
  response.bodyHandler(body -> {
    // Handle body entirely
  });
});
request.exceptionHandler(err -> {
  // Handle connection error OR response error
});
request.end();
```

以下の例は、Eclipse Vert.x 3.x リリースおよび Eclipse Vert.x 4 リリースで、アプリケーションを Web クライアントに移行する方法を示しています。

```
client.get(80, "example.com", "/some-uri")
  .send(ar -> {
    if (ar.suceeded()) {
      HttpResponse<Buffer> response = ar.result();
      // Handle response
    }  else {
      // Handle error
    }
  });
```

4.6.2.2. アプリケーションの Eclipse Vert.x HTTP クライアントへの移行

HTTP クライアントは、HTTP の対話を詳細に制御し、HTTP プロトコルに焦点をあてます。

Eclipse Vert.x 4 で HTTP クライアントが更新され、改善されました。

- より少ない対話による簡素化された API
- 強固なエラー処理
- HTTP/1 の接続リセットのサポート

HTTP クライアント API の更新は以下のとおりです。

- **HttpClientRequest** で、**get()**、**delete()**、**put()** などのメソッドが削除されました。代わりに **HttpClientRequest> request(HttpMethod method, ...)** メソッドを使用してください。
- 要求または応答が可能になると、**HttpClientRequest** インスタンスが作成されます。たとえば、**HttpClientRequest** インスタンスは、クライアントがサーバーに接続するか、プールから接続を再利用するときに作成されます。

4.6.2.2.1. シンプルな要求の送信

以下の例は、Eclipse Vert.x 3.x リリースで GET 要求を送信する方法を示しています。

```
HttpClientRequest request = client.get(80, "example.com", "/", response -> {
  int statusCode = response.statusCode();
  response.exceptionHandler(err -> {
    // Handle connection error, for example, connection closed
  });
```

```
  response.bodyHandler(body -> {
    // Handle body entirely
  });
});
request.exceptionHandler(err -> {
  // Handle connection error OR response error
});
request.end();
```

以下の例は、Eclipse Vert.x 4 で GET 要求を送信する方法を示しています。

```
client.request(HttpMethod.GET, 80, "example.com", "/", ar -> {
  if (ar.succeeded()) {
    HttpClientRequest = ar.result();
    request.send(ar2 -> {
      if (ar2.succeeded()) {
        HttpClientResponse = ar2.result();
        int statusCode = response.statusCode();
        response.body(ar3 -> {
          if (ar3.succeeded()) {
            Buffer body = ar3.result();
            // Handle body entirely
          } else {
            // Handle server error, for example, connection closed
          }
        });
      } else {
        // Handle server error, for example, connection closed
      }
    });
  } else {
    // Connection error, for example, invalid server or invalid SSL certificate
  }
});
```

新しい HTTP クライアントでエラー処理が優れていることが分かります。

以下の例は、Eclipse Vert.x 4 の GET 操作で future 構成を使用する方法を示しています。

```
Future<Buffer> fut = client.request(HttpMethod.GET, 80, "example.com", "/")
  .compose(request -> request.send().compose(response -> {
    int statusCode = response.statusCode();
    if (statusCode == 200) {
      return response.body();
    } else {
      return Future.failedFuture("Unexpectd status code");
    }
  })
});
fut.onComplete(ar -> {
  if (ar.succeeded()) {
    Buffer body = ar.result();
    // Handle body entirely
  } else {
```

```
    // Handle error
  }
});
```

今後の構成により、例外処理が改善されます。この例では、ステータスコードが 200 かどうかを確認し、200 でないとエラーを返します。

警告

future で HTTP クライアントを使用すると、**HttpClientResponse()** メソッドは応答を受信するとすぐにバッファーを生成し始めます。これを回避するには、(例にあるように) event-loop で future の構成が生じるか、応答を一時停止して再開する必要があります。

4.6.2.2.2. リクエストの送信

Eclipse Vert.x 3.x リリースでは、**end()** メソッドを使用して要求を送信できます。

```
request.end();
```

要求にボディーを送信することもできます。

```
request.end(Buffer.buffer("hello world));
```

HttpClientRequest は **Writestream<Buffer>** であるため、要求をストリーミングするのにパイプを使用することもできます。

```
writeStream.pipeTo(request, ar -> {
  if (ar.succeeded()) {
    // Sent the stream
  }
});
```

Eclipse Vert.x 4 では、**get()** メソッドを使用して例に示されるすべての操作を実行できます。新しい **send()** メソッドを使用してこれらの操作を実行することもできます。**send()** メソッドへの入力として、バッファー、文字列、または **ReadStream** を渡すことができます。このメソッドは、**HttpClientResponse** インスタンスを返します。

```
// Send a request and process the response
request.onComplete(ar -> {
  if (ar.succeeded()) {
    HttpClientResponse response = ar.result();
    // Handle the response
  }
})
request.end();

// The new send method combines all the operations
request.send(ar -> {
```

```
  if (ar.succeeded()) {
    HttpClientResponse response = ar.result();
    // Handle the response
  }
}));
```

4.6.2.2.3. 応答の処理

HttpClientResponse インターフェースが更新され、以下の方法で改善されました。

body() メソッド

body() メソッドは非同期バッファーを返します。**bodyHandler()** の代わりに **body()** メソッドを使用します。
以下の例は、**bodyHandler()** メソッドを使用して要求ボディーを取得する方法を示しています。

```
response.bodyHandler(body -> {
  // Process the request body
});
response.exceptionHandler(err -> {
  // Could not get the request body
});
```

以下の例は、**body()** メソッドを使用して要求ボディーを取得する方法を示しています。

```
response.body(ar -> {
  if (ar.succeeded()) {
    // Process the request body
  } else {
    // Could not get the request body
  }
});
```

end() メソッド

end() メソッドは、応答が正常または失敗した場合に future の結果を返します。このメソッドは応答ボディーを削除します。**endHandler()** メソッドの代わりにこのメソッドを使用してください。
以下の例は、**endHandler()** メソッドを使用する方法を示しています。

```
response.endHandler(v -> {
  // Response ended
});
response.exceptionHandler(err -> {
  // Response failed, something went wrong
});
```

以下の例は、**end()** メソッドの使用方法を示しています。

```
response.end(ar -> {
  if (ar.succeeded()) {
    // Response ended
  } else {
    // Response failed, something went wrong
  }
});
```

onSucces()、**compose()**、**bodyHandler()** などのメソッドで応答を処理することもできます。以下の例は、**onSuccess()** メソッドを使用して応答を処理する方法を示しています。

以下の例は、Eclipse Vert.x 3.x リリースにおいて、**result()** メソッドで HTTP クライアントを使用する方法を示しています。

```
HttpClient client = vertx.createHttpClient(options);

  client.request(HttpMethod.GET, 8443, "localhost", "/")
    .onSuccess(request -> {
      request.onSuccess(resp -> {

      //Code to handle HTTP response
      });
    });
```

以下の例は、Eclipse Vert.x 4 の **result()** メソッドで HTTP クライアントを使用する方法を示しています。

```
HttpClient client = vertx.createHttpClient(options);

  client.request(HttpMethod.GET, 8443, "localhost", "/")
    .onSuccess(request -> {
      request.response().onSuccess(resp -> {

      //Code to handle HTTP response
      });
    });
```

4.6.2.3. Eclipse Vert.x HTTP クライアントの改善

本セクションでは、HTTP クライアントの改善を説明します。

4.6.2.3.1. HTTP クライアント要求と応答メソッドが非同期ハンドラーを入力引数として取る

HttpClient メソッドおよび **HttpClientRequest** メソッドが更新され、非同期ハンドラーが使用されるようになりました。このメソッドは、**Handler<HttpClientResponse>** の代わりに、**Handler<AsyncResult<HttpClientResponse>>** を入力として取ります。

以前のリリースの Eclipse Vert.x では、リクエストを実行するためにさらに送信する必要がある **HttpClientRequest** を返すのに **HttpClient** メソッドの **getNow()**、**optionsNow()**、および **headNow()** が使用されていました。**getNow()** メソッド、**optionsNow()** メソッド、および **headNow()** メソッドが削除されました。Eclipse Vert.x 4 では、**Handler<AsyncResult<HttpClientResponse>>** を使用して、必要な情報で要求を直接送信できます。

以下の例は、Eclipse Vert.x 3.x で要求を送信する方法を示しています。

- GET 操作を実行するには、以下を実行します。

  ```
  Future<HttpClientResponse> f1 = client.get(8080, "localhost", "/uri", HttpHeaders.set("foo", "bar"));
  ```

- バッファーボディーを持つ POST を行うには、以下を行います。

```
Future<HttpClientResponse> f2 = client.post(8080, "localhost", "/uri", HttpHeaders.set("foo", "bar"), Buffer.buffer("some-data"));
```

- ストリーミングボディーを使用した POST を行うには、次のコマンドを実行します。

```
Future<HttpClientResponse> f3 = client.post(8080, "localhost", "/uri", HttpHeaders.set("foo", "bar"), asyncFile);
```

Eclipse Vert.x 4 では、**requests** メソッドを使用して **HttpClientRequest** インスタンスを作成できます。これらのメソッドは、以下のような基本的な対話で使用できます。

- リクエストヘッダーの送信
- プッシュハンドラー、ストリームの優先度、ping の設定などの HTTP/2 固有の操作。
- NetSocket トンネルの作成
- 粒度の細かい書き込み制御を提供
- ストリームのリセット
- 100 継続ヘッダーの手動処理

以下の例は、Eclipse Vert.x 4 で **HTTPClientRequest** を作成する方法を示しています。

```
client.request(HttpMethod.GET, 8080, "example.com", "/resource", ar -> {
  if (ar.succeeded()) {
    HttpClientRequest request = ar.result();
    request.putHeader("content-type", "application/json")
    request.send(new JsonObject().put("hello", "world"))
      .onSuccess(response -> {
      //
      }).onFailure(err -> {
      //
       });
     }
})
```

4.6.2.3.2. HTTP クライアント要求から接続ハンドラーメソッドを削除

HttpClientRequest.connectionHandler() メソッドが削除されました。代わりに **HttpClient.connectionHandler()** メソッドを使用して、アプリケーションのクライアントリクエストの接続ハンドラーを呼び出します。

以下の例は、**HttpClientRequest.connectionHandler()** メソッドが Eclipse Vert.x 3.x リリースでどのように使用されたかを示しています。

```
client.request().connectionHandler(conn -> {
  // Connection related code
}).end();
```

以下の例は、新しい **HttpClient.connectionHandler()** メソッドを使用する方法を表しています。

```
client.connectionHandler(conn -> {
// Connection related code
});
```

4.6.2.3.3. net ソケットメソッドを使用した HTTP クライアントトンネリング

HTTP トンネルは、**HttpClientResponse.netSocket()** メソッドを使用して作成できます。Eclipse Vert.x 4 では、このメソッドが更新されました。

要求の接続に net ソケットを取得するには、要求でソケットハンドラーを送信します。ハンドラーは、HTTP 応答ヘッダーの受信時に呼び出されます。ソケットはトンネリングの準備ができ、バッファーの送受信が可能です。

以下の例は、Eclipse Vert.x 3.x リリースで接続の net ソケットを取得する方法を示しています。

```
client.request(HttpMethod.CONNECT, uri, ar -> {
  if (ar.succeeded()) {
    HttpClientResponse response = ar.result();
    if (response.statusCode() == 200) {
      NetSocket so = response.netSocket();
   }
  }
}).end();
```

以下の例は、Eclipse Vert.x 4 で接続の net ソケットを取得する方法を示しています。

```
client.request(HttpMethod.CONNECT, uri, ar -> {
}).netSocket(ar -> {
  if (ar.succeeded()) {
   // Got a response with a 200 status code
   NetSocket so = ar.result();
   // Go for tunneling
  }
}).end();
```

4.6.2.3.4. HttpClient クラスの新しい send() メソッド

HttpClient クラスで新しい **send()** メソッドを利用可能

以下のコードは、Eclipse Vert.x 4 で要求を送信する方法を示しています。

```
Future<HttpClientResponse> f1 = client.send(HttpMethod.GET, 8080, "localhost", "/uri",
HttpHeaders.set("foo", "bar"));
```

4.6.2.3.5. HttpHeaders はインターフェースで、MultiMap メソッドを含む

Eclipse Vert.x 4 では、**HttpHeaders** はインターフェースです。以前のリリースの Eclipse Vert.x では、**HttpHeaders** はクラスでした。

以下の新しい **MultiMap** メソッドが **HttpHeaders** インターフェースに追加されました。これらのメソッドを使用して **MultiMap** インスタンスを作成します。

- **MultiMap.headers()**

- **MultiMap.set(CharSequence name, CharSequence value)**
- **MultiMap.set(String name, String value)**

以下の例は、Eclipse Vert.x 3.x リリースで **MultiMap** インスタンスが作成された方法を示しています。

```
MultiMap headers = MultiMap.caseInsensitiveMultiMap();
```

以下の例は、Eclipse Vert.x 4 で **MultiMap** インスタンスを作成する方法を示しています。

```
MultiMap headers = HttpHeaders.headers();
```

```
MultiMap headers = HttpHeaders.set("content-type", "application.data");
```

4.6.2.3.6. **CaseInsensitiveHeaders** クラスが公開されなくなる

CaseInsensitiveHeaders クラスが公開されなくなりました。**MultiMap.caseInsensitiveMultiMap()** メソッドを使用して、大文字と小文字を区別しないキーを持つマルチマップ実装を作成します。

以下の例は、Eclipse Vert.x 3.x リリースで **CaseInsensitiveHeaders** メソッドがどのように使用されたかを示しています。

```
CaseInsensitiveHeaders headers = new CaseInsensitiveHeaders();
```

以下の例は、Eclipse Vert.x 4 で **MultiMap** メソッドを使用する方法を示しています。

```
MultiMap multiMap = MultiMap#caseInsensitiveMultiMap();
```

あるいは

```
MultiMap headers = HttpHeaders.headers();
```

4.6.2.3.7. サーバーで実行している **HTTP** のバージョンの確認

以前のリリースの Eclipse Vert.x では、**HttpServerRequest.version()** メソッドを明示的に呼び出している場合に限り、サーバーで実行している HTTP のバージョンががチェックされました。HTTP バージョンが HTTP/1.x の場合、メソッドは 501 HTTP ステータスを返し、接続を閉じます。

Eclipse Vert.x 4 以降、要求がサーバーに送信される前に、サーバー上の HTTP バージョンは **HttpServerRequest.version()** メソッドを呼び出して自動的にチェックされます。このメソッドは、無効な HTTP バージョンが見つかったときに例外を出力する代わりに、HTTP バージョンを返します。

4.6.2.3.8. 要求オプションの新しいメソッド

Eclipse Vert.x 4 では、以下の新しいメソッドが **RequestOptions** クラスで利用できます。

- Header
- FollowRedirects
- Timeout

以下の例は、新しいメソッドの使用例を示しています。

```
client.request(HttpMethod.GET, 8080, "example.com", "/resource", ar -> {
  if (ar.succeeded()) {
    HttpClientRequest request = ar.result();
    request.putHeader("content-type", "application/json")
    request.send(new JsonObject().put("hello", "world"))
      .onSuccess(response -> {
      //
      }).onFailure(err -> {
      //
       });
     }
})
```

4.7. 接続メソッドの変更点

本セクションでは、接続方法の変更点を説明します。

4.7.1. クライアントに認証が必要であるかどうかの確認

NetServerOptions.isClientAuthRequired() メソッドが削除されました。**getClientAuth() == ClientAuth.REQUIRED** 列挙タイプを使用して、クライアント認証が必要であるかどうかを確認します。

以下の例は、switch ステートメントを使用して、クライアントの認証が必要であるかどうかを確認する方法を示しています。

```
switch (options.getClientAuth()) {
case REQUIRED:
// ... behavior same as in releases prior to {VertX} {v4}
break;
default:
// fallback statement...
}
```

以下の例は、Eclipse Vert.x 4 でクライアントの認証が必要な場合にチェックを使用する方法を示しています。

```
if (options.getClientAuth() == ClientAuth.REQUIRED) {
// behavior in releases prior to {VertX} {v4}
```

4.7.2. アップグレード SSL メソッドによる非同期ハンドラーの使用

NetSocket.upgradeToSsl() メソッドが更新され、**Handler** ではなく **Handler<AsyncResult>** が使用されるようになりました。ハンドラーは、チャネルが SSL または TLS に正常にアップグレードされているかどうかを確認するために使用されます。

4.8. ロギングの変更

本セクションでは、ロギングの変更点を説明します。

4.8.1. 非推奨のロギングクラスおよびメソッド

ロギングクラス **Logger** および **LoggerFactory** と、それらのメソッドが非推奨になりました。これらのロギングクラスおよびメソッドは今後のリリースで削除されます。

4.8.2. 削除された Log4j 1 ロガー

Log4j 1 ロガーは利用できなくなりました。しかし、**Log4j 1** ロガーを使用する場合は **SLF4J** で利用できます。

4.9. ECLIPSE VERT.X REACTIVE EXTENSIONS (RX) の変更点

このセクションでは、Eclipse Vert.x の Reactive Extensions (Rx) の変更点を説明します。Eclipse Vert.x は RxJava ライブラリーを使用します。

4.9.1. 書き込みストリームから、**onComplete** コールバックを削除

WriteStreamSubscriber.onComplete() コールバックが削除されました。このコールバックは、**WriteStream** が書き込まれるデータのストリームが保留中の場合に呼び出されました。

Eclipse Vert.x 4 では、代わりに **WriteStreamSubscriber.onWriteStreamEnd()** コールバックおよび **WriteStreamSubscriber.onWriteStreamError()** コールバックを使用します。これらのコールバックは、**WriteStream.end()** の完了後に呼び出されます。

```
WriteStreamSubscriber<Buffer> subscriber = writeStream.toSubscriber();
```

以下の例は、Eclipse Vert.x 3.x リリースの **WriteStream** からアダプターを作成する方法を示しています。

```
subscriber.onComplete(() -> {
  // Called after writeStream.end() is invoked, even if operation has not completed
});
```

以下の例は、Eclipse Vert.x 4 リリースの新しいコールバックメソッドを使用して WriteStream からアダプターを作成する方法を示しています。

```
subscriber.onWriteStreamEnd(() -> {
  // Called after writeStream.end() is invoked and completes successfully
});
```

```
subscriber.onWriteStreamError(() -> {
  // Called after writeStream.end() is invoked and fails
});
```

4.10. ECLIPSE VERT.X 設定の変更点

以下のセクションでは、Eclipse Vert.x 設定の変更点を説明します。

4.10.1. 設定を取得する新しい方法

ConfigRetriever.getConfigAsFuture() メソッドが削除されました。代わりに、**retriever.getConfig()** メソッドを使用してください。

以下の例は、Eclipse Vert.x 3.x リリースで設定を取得する方法を示しています。

```
Future<JsonObject> fut = ConfigRetriever. getConfigAsFuture(retriever);
```

以下の例は、Eclipse Vert.x 4 で設定を取得する方法を示しています。

```
fut = retriever.getConfig();
```

4.11. JSON の変更点

本セクションでは、JSON の変更点を説明します。

4.11.1. Jackson のカプセル化

Jackson 型を実装する JSON クラスのメソッドがすべて削除されました。代わりに以下の方法を使用してください。

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)**

たとえば、以下のコードを使用します。

- Jackson の **TypeReference** を使用する場合:
  - Eclipse Vert.x 3.x リリースの場合:

    ```
    List<Foo> foo1 = Json.decodeValue(json, new TypeReference<List<Foo>>() {});
    ```

  - Eclipse Vert.x 4 リリースの場合:

    ```
    List<Foo> foo2 = io.vertx.core.json.jackson.JacksonCodec.decodeValue(json, new TypeReference<List<Foo>>() {});
    ```

- **ObjectMapper**の参照:
  - Eclipse Vert.x 3.x リリースの場合:

    ```
    ObjectMapper mapper = Json.mapper;
    ```

  - Eclipse Vert.x 4 リリースの場合:

    ```
    mapper = io.vertx.core.json.jackson.DatabindCodec.mapper();
    ```

- **ObjectMapper** の設定:
  - Eclipse Vert.x 3.x リリースの場合:

    ```
    Json.mapper = someMapper;
    ```

  - Eclipse Vert.x 4 以降では、mapper インスタンスを作成することはできません。独自の静的マッパーを使用するか、**Databind.mapper()** インスタンスを設定する必要があります。

4.11.2. オブジェクトマッピング

以前のリリースでは、Jackson コアおよび Jackson データバインドの依存関係がランタイム時に必要でした。

Eclipse Vert.x 4 以降では、Jackson のコア依存関係のみが必要になります。

オブジェクトマッピング JSON の場合のみ、Jackson データバインドの依存関係が必要になります。この場合は、**com.fasterxml.jackson.core:jackson-databind** jar のプロジェクト記述子に依存関係を明示的に追加する必要があります。

以下のメソッドは上記の型でサポートされます。

- メソッド
  - **JsonObject.mapFrom(Object)**
  - **JsonObject.mapTo(Class)**
  - **Json.decodeValue(Buffer, Class)**
  - **Json.decodeValue(String, Class)**
  - **Json.encode(Object)**
  - **Json.encodePrettily(Object)**
  - **Json.encodeToBuffer(Object)**
- 型
  - **JsonObject** および **JsonArray**
  - **Map** および **List**
  - **Number**
  - **Boolean**
  - **Enum**
  - **byte[]** および **Buffer**
  - **Instant**

以下のメソッドは、Jackson バインドでのみサポートされます。

- **JsonObject.mapTo(Object)**

- **JsonObject.mapFrom(Object)**

4.11.3. Base64 エンコーダーを JSON オブジェクトおよびアレイの Base64URL に更新

Eclipse Vert.x の JSON 型は RFC-7493 を実装します。以前のリリースの Eclipse Vert.x では、実装は Base64URL ではなく Base64 エンコーダーが誤って使用されていました。Eclipse Vert.x 4 で修正され、JSON 型で Base64URL エンコーダーが使用されました。

Eclipse Vert.x 4 で Base64 エンコーダーを引き続き使用する場合は、設定フラグの **legacy** を使用できます。以下の例は、Eclipse Vert.x 4 で設定フラグを設定する方法を示しています。

```
java -Dvertx.json.base64=legacy ...
```

新しいユーティリティーメソッド **Json.toBase64()** は Base64URL 文字列を Base64 に変換します。このユーティリティーは、以下を可能にします。

- Eclipse Vert.x 3.x リリースから Eclipse Vert.x 4 への移行中に統合の処理。
- Base64 文字列を使用する他のシステムとの相互運用性の処理。

以下のコード例を使用して、Base64URL を Base64 エンコーダーに変換します。

```
String base64url = someJsonObject.getString("base64encodedElement")
String base64 = Json.toBase64(base64url);
```

4.11.4. 信頼オプションから JSON コンバーターメソッドを削除

TrustOptions.toJSON メソッドが削除されました。

4.12. ECLIPSE VERT.X WEB の変更点

以下のセクションでは、Eclipse Vert.x Web の変更点を説明します。

4.12.1. セッションハンドラーでのユーザーセッションハンドラーの機能の組み合わせ

以前のリリースの Eclipse Vert.x では、セッション内で動作しているときに **UserSessionHandler** ハンドラーと **SessionHandler** ハンドラーの両方を指定する必要がありました。

プロセスを単純化するために、Eclipse Vert.x 4 では **UserSessionHandler** クラスが削除され、その機能が **SessionHandler** クラスに追加されました。Eclipse Vert.x 4 では、セッションと連携するには 1 つのハンドラーのみを指定する必要があります。

4.12.2. Cookie インターフェースの削除

以下の Cookie インターフェースが削除されました。

- **io.vertx.ext.web.Cookie**
- **io.vertx.ext.web.handler.CookieHandler**

代わりに **io.vertx.core.http.Cookie** インターフェースを使用してください。

4.12.3. Favicon およびエラーハンドラーが Vertx ファイルシステムを使用

FaviconHandler および **ErrorHandler** で create メソッドが更新されました。create メソッドで **Vertx** インスタンスオブジェクトを渡す必要があります。これらのメソッドはファイルシステムにアクセスします。**Vertx** オブジェクトを渡すと、'Vertx' ファイルシステムを使用するファイルへのアクセスの一貫性が確保されます。

以下の例は、Eclipse Vert.x 3.x リリースで create メソッドを使用する方法を示しています。

```
FaviconHandler.create();
ErrorHandler.create();
```

以下の例は、Eclipse Vert.x 4 で create メソッドを使用する方法を示しています。

```
FaviconHandler.create(vertx);
ErrorHandler.create(vertx);
```

4.12.4. テンプレートエンジンへのアクセス

TemplateEngine.unwrap() メソッドを使用してテンプレートエンジンにアクセスします。その後、カスタマイズおよび設定をテンプレートに適用できます。

エンジン設定の取得および設定に使用される以下のメソッドは非推奨になりました。代わりに **TemplateEngine.unwrap()** メソッドを使用します。

- **HandlebarsTemplateEngine.getHandlebars()**
- **HandlebarsTemplateEngine.getResolvers()**
- **HandlebarsTemplateEngine.setResolvers()**
- **JadeTemplateEngine.getJadeConfiguration()**
- **ThymeleafTemplateEngine.getThymeleafTemplateEngine()**
- **ThymeleafTemplateEngine.setMode()**

4.12.5. ロケールインターフェースを削除

io.vertx.ext.web.Locale インターフェースが削除されました。代わりに **io.vertx.ext.web.LanguageHeader** インターフェースを使用してください。

4.12.6. 使用できるロケールメソッドを削除

RoutingContext.acceptableLocales() メソッドが削除されました。代わりに **RoutingContext.acceptableLanguages()** メソッドを使用してください。

4.12.7. サブルーターをマウントする方法を更新

以前のリリースの Eclipse Vert.x では、**Router.mountSubRouter()** メソッドが **Router** を誤って返していました。これは修正され、メソッドが **Route** を返すようになりました。

4.12.8. JWT 認証処理の除外文字列を使用して create メソッドを削除

JWTAuthHandler.create(JWTAuth authProvider, String skip) メソッドが削除されました。代わりに **JWTAuthHandler.create(JWTAuth authProvider)** メソッドを使用します。

以下の例は、Eclipse Vert.x 3.x リリースで JWT 認証ハンドラーが作成された方法を表しています。

```
router
  // protect everything but "/excluded/path"
  .route().handler(JWTAuthHandler(jwtAuth, "/excluded/path")
```

以下の例は、Eclipse Vert.x 4 で JWT 認証ハンドラーが作成された方法を表しています。

```
router
  .route("/excluded/path").handler(/* public access to "/excluded/path" */)
  // protect everything
  .route().handler(JWTAuthHandler(jwtAuth)
```

4.12.9. OSGi 環境で使用される create ハンドラーメソッドが削除

Eclipse Vert.x 4 では、OSGi 環境はサポートされなくなりました。**StaticHandler.create(String, ClassLoader)** メソッドは OSGi 環境内で使用されたため、削除されました。

アプリケーションでこの方法を使用した場合は、Eclipse Vert.x 4 でリソースをアプリケーションクラスパスに追加するか、ファイルシステムからリソースを提供できます。

4.12.10. ブリッジオプションクラスを削除

sockjs.BridgeOptions クラスが削除されました。代わりに新しい **sockjs.SockJSBridgeOptions** クラスを使用してください。**sockjs.SockJSBridgeOptions** クラスには、イベントバスブリッジの設定に必要なすべてのオプションが含まれます。

データオブジェクトクラスの名前が変更された場合を除き、新しいクラスの動作は変更されません。

以前のリリースでは、**sockjs.BridgeOptions** クラスを使用して新規ブリッジを追加すると、多くの重複した設定がありました。新しいクラスには可能な共通設定がすべて含まれ、重複した設定を削除します。

4.12.11. SockJS ソケットイベントバスが、デフォルトでクラスターイベントを登録しない

SockJSSocket は、デフォルトでクラスター化されたイベントバスコンシューマーを登録しなくなりました。イベントバスを使用してソケットに書き込む場合は、**SockJSHandlerOptions** で **writeHandler** を有効にする必要があります。**writeHandler** を有効にすると、イベントバスコンシューマーはデフォルトでローカルに設定されます。

```
Router router = Router.router(vertx);
SockJSHandlerOptions options = new SockJSHandlerOptions()
  .setRegisterWriteHandler(true); // enable the event bus consumer registration
SockJSHandler sockJSHandler = SockJSHandler.create(vertx, options);
router.mountSubRouter("/myapp", sockJSHandler.socketHandler(sockJSSocket -> {
  // Retrieve the writeHandlerID and store it (For example, in a local map)
  String writeHandlerID = sockJSSocket.writeHandlerID();
}));
```

イベントバスコンシューマーをクラスターに設定できます。

```
SockJSHandlerOptions options = new SockJSHandlerOptions()
  .setRegisterWriteHandler(true) // enable the event bus consumer registration
  .setLocalWriteHandler(false) // register a clustered event bus consumer
```

4.12.12. 認証プロバイダーを追加する新しい方法

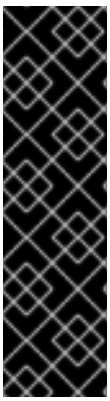
SessionHandler.setAuthProvider(AuthProvider) メソッドが非推奨になりました。代わりに **SessionHandler.addAuthProvider()** メソッドを使用してください。新しいメソッドにより、アプリケーションは複数の認証プロバイダーと連携でき、セッションオブジェクトをこれらの認証プロバイダーにリンクできます。

4.12.13. OAuth2 認証プロバイダーの作成メソッドにはコンストラクター引数として vertx が必要

Eclipse Vert.x 4 以降では、**OAuth2Auth.create(Vertx vertx)** メソッドにコンストラクターの引数として **vertx** が必要です。**vertx** 引数は、安全な非ブロッキングの乱数ジェネレーターを使用して nonce を生成するため、アプリケーションのセキュリティーが向上します。

4.13. ECLIPSE VERT.X WEB GRAPHQL の変更点

以下のセクションでは、Eclipse Vert.x Web GraphQL の変更点を説明します。



重要

Eclipse Vert.x Web GraphQL はテクノロジープレビューとしてのみ提供されます。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。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。

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「テクノロジプレビュー機能のサポート範囲」を参照してください。

4.13.1. 複数の言語 (polyglot) 環境でサポートされる更新メソッド

以下のメソッドが更新され、polyglot 環境でサポートされるようになりました。* **UploadScalar** がファクトリーとなりました。代わりに **UploadScalar.create()** メソッドを使用してください。

- **VertxBatchLoader** がファクトリーとなりました。代わりに **io.vertx.ext.web.handler.graphql.dataloader.VertxBatchLoader.create()** メソッドを使用してください。
- **VertxDataFetcher** がファクトリーとなりました。代わりに **io.vertx.ext.web.handler.graphql.schema.VertxDataFetcher.create()** メソッドを使用してください。
- **VertxPropertyDataFetcher** がファクトリーとなりました。代わりに **io.vertx.ext.web.handler.graphql.schema.VertxPropertyDataFetcher.create()** メソッドを使用してください。

4.13.2. Eclipse Vert.x Web GraphQL での POST 要求の処理

本リリース以前は、Eclipse Vert.x Web GraphQL ハンドラーが独自の POST 要求を処理する可能性がありました。要求を処理するのに Eclipse Vert.x Web **BodyHandler** は必要ありませんでした。しかし、この実装は DDoS 攻撃の影響を受けやすくなりました。

Eclipse Vert.x 4 以降から POST 要求を処理するには、**BodyHandler** が必要になります。Eclipse Vert.x Web GraphQL ハンドラーをインストールする前に、**BodyHandler** をインストールする必要があります。

4.14. MICROMETER メトリクスの変更

以下のセクションでは、Micmeter メトリクスの変更点を説明します。

4.14.1. TCP の送受信バイト数は、同等の HTTP 要求と応答サマリーを持つカウンターとして記録

本リリース以前は、以下のメトリクスはソケットのディストリビューションサマリーとして記録されていました。Eclipse Vert.x 4 以降では、これらのメトリクスはカウンターとしてログに記録され、交換されたデータ量を報告します。

- Net クライアント
  - **vertx_net_client_bytes_read**
  - **vertx_net_client_bytes_written**
- Net サーバー
  - **vertx_net_server_bytes_read**
  - **vertx_net_server_bytes_written**

これらのカウンターでは、HTTP 用に同等のディストリビューションサマリーが導入されました。これらのサマリーは、要求および応答のサイズに関する情報を収集するために使用されます。

- HTTP クライアント
  - **vertx_http_client_request_bytes**
  - **vertx_http_client_response_bytes**
- HTTP サーバー
  - **vertx_http_server_request_bytes**
  - **vertx_http_server_response_bytes**

4.14.2. メトリクスの名前変更

以下のメトリクスの名前が変更されました。

旧メトリクス名	新メトリクス名	コンポーネントの更新

旧メトリクス名	新メトリクス名	コンポーネントの更新
***_connections**	***_active_connections**	Net クライアントおよびサーバー HTTP クライアントおよびサーバー
***_bytesReceived**	***_bytes_read**	データグラム Net クライアントおよびサーバー HTTP クライアントおよびサーバー
***_bytesSent**	***_bytes_written**	データグラム Net クライアントおよびサーバー HTTP クライアントおよびサーバー
***_requests**	***_active_requests**	HTTP クライアント HTTP サーバー
***_requestCount_total**	***_requests_total**	HTTP クライアント HTTP サーバー
***_responseTime_seconds**	***_response_time_seconds**	HTTP クライアント HTTP サーバー
***_responseCount_total**	***_responses_total**	HTTP クライアント HTTP サーバー
***_wsConnections**	***_active_ws_connections**	HTTP クライアント HTTP サーバー
vertx_http_client_queue_delay_seconds	**vertx_http_client_queue_time_seconds**	
vertx_http_client_queue_size	**vertx_http_client_queue_pending**	
vertx_http_server_requestResetCount_total	**vertx_http_server_request_resets_total**	
vertx_eventbus_bytesWritten	**vertx_eventbus_bytes_written**	

旧メトリクス名	新メトリクス名	コンポーネントの更新
vertx_eventbus_bytesRead	**vertx_eventbus_bytes_read**	
vertx_eventbus_replyFailures	**vertx_eventbus_reply_failures**	
vertx_pool_queue_delay_seconds	**vertx_pool_queue_time_seconds**	
vertx_pool_queue_size	**vertx_pool_queue_pending**	
vertx_pool_inUse	**vertx_pool_in_use**	

4.15. ECLIPSE VERT.X OPENAPI の変更点

Eclipse Vert.x 4 では、新しいモジュール **vertx-web-openapi** が利用できるようになりました。このモジュールは、**vertx-web** と併用して、コントラクト駆動型アプリケーションを開発します。

新しいモジュールは、Eclipse Vert.x Web **Router** と適切に機能します。新しいモジュールには以下の Eclipse Vert.x 依存関係が必要です。

- **vertx-json-schema**
- **vertx-web-validation**

新しいモジュールは、**io.vertx.ext.web.openapi** パッケージで利用できます。

Eclipse Vert.x 4 では、新しいモジュールへの移行を容易にするために、古い OpenAPI モジュール **vertx-web-api-contract** がサポートされます。新しい機能を活用するには、新しいモジュール **vertx-web-openapi** を使用することが推奨されます。

4.15.1. 新規モジュールはルータービルダーを使用

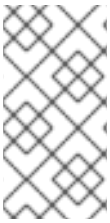
vertx-web-openapi モジュールは、**RouterBuilder** を使用して Eclipse Vert.x Web ルーターをビルドします。このルータービルダーは、**vertx-web-api-contract** モジュールのルータービルダー **OpenAPI3RouterFactory** と似ています。

vertx-web-openapi モジュールの使用を開始するには、**RouterBuilder** をインスタンス化します。

```
RouterBuilder.create(vertx, "petstore.yaml").onComplete(ar -> {
  if (ar.succeeded()) {
    // Spec loaded with success
    RouterBuilder routerBuilder = ar.result();
  } else {
    // Something went wrong during router builder initialization
    Throwable exception = ar.cause();
  }
});
```

future を使用して **RouterBuilder** をインスタンス化することもできます。

```
RouterBuilder.create(vertx, "petstore.yaml")
  .onSuccess(routerBuilder -> {
    // Spec loaded with success
  })
  .onFailure(exception -> {
    // Something went wrong during router builder initialization
  });
```



注記

vertx-web-openapi モジュールは、Eclipse Vert.x ファイルシステム API を使用してファイルを読み込みます。そのため、クラスパスリソースに / を指定する必要はありません。たとえば、アプリケーションに **petstore.yaml** を指定することができます。**RouterBuilder** は、クラスパスリソースからコントラクトを特定できます。

4.15.2. 新たなルータービルダーメソッド

ほとんどの場合、古い **OpenAPI3RouterFactory** メソッドを検索して、新しい **RouterBuilder** メソッドに置き換えることができます。以下の表は、旧式のメソッドと新しいメソッドの例をいくつか紹介します。

古い OpenAPI3RouterFactory メソッド	新しい RouterBuilder メソッド
routerFactory.addHandlerByOperationId("getPets", handler)	**routerBuilder.operation("getPets").handler(handler)**
routerFactory.addFailureHandlerByOperationId("getPets", handler)	**routerBuilder.operation("getPets").failureHandler(handler)**
routerFactory.mountOperationToEventBus("getPets", "getpets.myapplication")	**routerBuilder.operation("getPets").routeToEventBus("getpets.myapplication")**
routerFactory.addGlobalHandler(handler)	**routerBuilder.rootHandler(handler)**
routerFactory.addBodyHandler(handler)	**routerBuilder.bodyHandler(handler)**
routerFactory.getRouter()	**routerBuilder.createRouter()**

以下の構文を使用して、解析された要求パラメーターにアクセスします。

```
RequestParameters parameters =
routingContext.get(io.vertx.ext.web.validation.ValidationHandler.REQUEST_CONTEXT_KEY);
int aParam = parameters.queryParameter("aParam").getInteger();
```

4.15.3. セキュリティーの処理

Eclipse Vert.x 4 では、**RouterFactory.addSecurityHandler()** メソッドおよび **OpenAPI3RouterFactory.addSecuritySchemaScopeValidator()** メソッドは利用できなくなりました。

代わりに **RouterBuilder.securityHandler()** メソッドを使用します。このメソッドは、**io.vertx.ext.web.handler.AuthenticationHandler** をハンドラーとして受け入れます。このメソッドは **OAuth2Handler** を自動的に認識し、セキュリティースキーマを設定します。

新しいセキュリティーハンドラーは、OpenAPI 仕様 で定義された操作も実装します。

4.15.4. 一般的な障害の処理

vertx-web-openapi モジュールでは、以下の失敗ハンドラーは利用できません。**Router.errorHandler(int, Handler)** メソッドを使用して失敗ハンドラーを設定する必要があります。

`vertx-web-api-contract` モジュールの旧メソッド	vertx-web-openapi モジュールの新しいメソッド
routerFactory.setValidationFailureHandler(handler)	**router.errorHandler(400, handler)**
routerBuilder.setNotImplementedFailureHandler(handler)	**router.errorHandler(501, handler)**

4.15.5. OpenAPI 契約モデルへのアクセス

Eclipse Vert.x 4 では、OpenAPI コントラクトは POJO (Plain Old Java Object) にマッピングされません。そのため、追加の swagger-parser 依存関係は不要になりました。ゲッターおよびリゾルバーを使用して、コントラクトの特定のコンポーネントを取得できます。

以下の例は、1 つの操作を使用して特定のコンポーネントを取得する方法を示しています。

```
JsonObject model = routerBuilder.operation("getPets").getOperationModel();
```

以下の例は、完全なコントラクトを取得する方法を示しています。

```
JsonObject contract = routerBuilder.getOpenAPI().getOpenAPI();
```

以下の例は、コントラクトの一部を解決する方法を示しています。

```
JsonObject petModel =
routerBuilder.getOpenAPI().getCached(JsonPointer.from("/components/schemas/Pet"));
```

4.15.6. OpenAPI を使用しない Web 要求の検証

vertx-web-api-contract モジュールでは、**HTTPRequestValidationHandler** を使用して HTTP リクエストを検証できます。検証に OpenAPI を使用する必要はありません。

Eclipse Vert.x 4 では、HTTP 要求を検証するため **vertx-web-validation** モジュールを使用します。このモジュールをインポートし、OpenAPI を使用せずにリクエストを検証できます。**ValidationHandler** を使用して要求を検証します。

4.15.7. Eclipse Vert.x Web API サービスの更新

vertx-web-api-service モジュールが更新され、**vertx-web-validation** モジュールとともに使用できるようになりました。**vertx-web-openapi** モジュールを使用している場合、Web サービス機能は変更されません。

ただし、OpenAPI を使用しない場合は、**vertx-web-validation** モジュールで Web サービスモジュールを使用するには、**RouteToEBServiceHandler** クラスを使用する必要があります。

```
router.get("/api/transactions")
  .handler(
    ValidationHandlerBuilder.create(schemaParser)
      .queryParameter(optionalParam("from", stringSchema()))
      .queryParameter(optionalParam("to", stringSchema()))
      .build()
  ).handler(
    RouteToEBServiceHandler.build(eventBus, "transactions.myapplication", "getTransactionsList")
  );
```

vertx-web-api-service モジュールは **vertx-web-api-contract** をサポートしません。そのため、Eclipse Vert.x 4 にアップグレードする場合は、Eclipse Vert.x OpenAPI アプリケーションを **vertx-web-openapi** モジュールに移行する必要があります。

第5章 マイクロサービスパターンの変更

本セクションでは、マイクロサービスパターンの変更点を説明します。

5.1. ECLIPSE VERT.X サーキットブレーカーの変更点

以下のセクションでは、Eclipse Vert.x サーキットブレーカーの変更点を説明します。

5.1.1. サーキットブレーカーで実行コマンドメソッドを削除

以下のメソッドは future で使用できないため、**CircuitBreaker** クラスから削除されました。

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()**

5.2. ECLIPSE VERT.X サービス検出の変更点

以下のセクションでは、Eclipse Vert.x サービス検出の変更点を説明します。

5.2.1. ServiceDiscovery 引数が含まれるサービス検出から create メソッドを削除

Handler<AmqpMessage> を引数として持つサービス検出で、以下の作成メソッドが削除されました。future では、以下のメソッドは使用できません。

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)**

5.2.2. サービスの importer および exporter メソッドが fluent でなくなる

ServiceDiscovery.registerServiceImporter() メソッドおよび **ServiceDiscovery.registerServiceExporter()** メソッドは fluent ではなくなりました。メソッドでは、**Future<Void>** が返されます。

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

vertx-service-discovery-bridge-kubernetes は **KubernetesServiceImporter** 検出ブリッジを追加します。ブリッジは、Kubernetes または Openshift から Eclipse Vert.x サービス検出にサービスをインポートします。

Eclipse Vert.x 4 以降、この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。

以下の例では、ブリッジを手動で登録する方法を説明します。

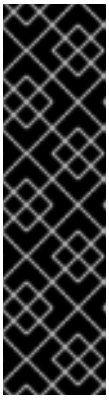
```
JsonObject defaultConf = new JsonObject();
serviceDiscovery.registerServiceImporter(new KubernetesServiceImporter(), defaultConf);
```

第6章 ECLIPSE VERT.X 認証および承認の変更点

以下のセクションでは、Eclipse Vert.x 認証および承認の変更点を説明します。

Eclipse Vert.x 認証モジュールには、Eclipse Vert.x 4 に主要な更新があります。**io.vertx.ext.auth.AuthProvider** インターフェースは、2 つの新しいインターフェースに分割されました。

- **io.vertx.ext.auth.authentication.AuthenticationProvider**



重要

認証機能は、テクノロジープレビューとしてのみ提供されます。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。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。

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「テクノロジプレビュー機能のサポート範囲」を参照してください。

- **io.vertx.ext.auth.authorization.AuthorizationProvider**

今回の更新で、プロバイダーは認証および承認のいずれかを個別に実行できるようになりました。

6.1. 認証アプリケーションの移行

認証メカニズムは結果レベルで変更になりました。以前のリリースでは、結果はプロバイダー固有の **User** オブジェクトでした。Eclipse Vert.x 4 では、結果は **io.vertx.ext.auth.User** の一般的な実装になります。

以下の例は、Eclipse Vert.x 3.x リリースでユーザーが認証された方法を表しています。

```
JsonObject authInfo = new JsonObject()
  .put("username", "john")
  .put("password", "super$ecret");

// omitting the error handling for brevity
provider.authenticate(authInfo, res -> {
  if (res.succeeded()) {
    // may require type casting for example on Oauth2
    User user = res.result();
  }
});
```

以下の例は、Eclipse Vert.x 4 でユーザーを認証する方法を示しています。

```
JsonObject authInfo = new JsonObject()
  .put("username", "john")
  .put("password", "super$ecret");

// omitting the error handling for brevity
provider.authenticate(authInfo, res -> {
```

```
  if (res.succeeded()) {
    // Never needs type casting
    User user = res.result();
  }
});
```

6.2. 承認アプリケーションの移行

承認は Eclipse Vert.x 4 の新機能です。以前のリリースでは、ユーザーが **User** オブジェクトでタスクを実行することが許可されているかどうかを確認できました。そのため、プロバイダーはユーザーの認証と承認の両方を行います。

Eclipse Vert.x 4 では、**User** オブジェクトインスタンスは特定の認証プロバイダーに関連付けられていません。そのため、異なるプロバイダーを使用してユーザーを認証および承認できます。たとえば、OAuth2 を使用してユーザーを認証し、MongoDB または SQL データベースに対して承認チェックを実行できます。

以下の例は、ユーザーが Eclipse Vert.x 3.x リリースでプリンター #1234 を使用できるかどうかについてアプリケーションをチェックする方法を示しています。

```
// omitting the error handling for brevity
user.isAuthorized("printers:printer1234", res -> {
  if (res.succeeded()) {
    boolean hasAuthority = res.result();
    if (hasAuthority) {
      System.out.println("User can use the printer");
    } else {
      System.out.println("User cannot use the printer");
    }
  }
});
```

この承認は JDBC および MongoDB で有効にしました。しかし、プロバイダーは承認チェックを実行しなかったため、OAuth2 などのプロバイダーでは機能しませんでした。Eclipse Vert.x 4 以降では、異なるプロバイダーを使用してこのような承認チェックを実行できます。

```
// omitting the error handling for brevity
provider.getAuthorizations(user, res -> {
  if (res.succeeded()) {
    if (PermissionBasedAuthorization.create("printer1234").match(user)) {
      System.out.println("User can use the printer");
    } else {
      System.out.println("User cannot use the printer");
    }
  }
});
```

承認は、ロール、パーミッション、ロジック操作、ワイルドカード、および追加するその他の実装で確認することができます。

6.3. キー管理の変更点

Eclipse Vert.x 4 では、キーの処理に主要な更新があります。最も重要な変更は、キーの読み込み時にパブリックバッファーとプライベートバッファーの区別がないことです。

以下のクラスが更新されました。

- **jce** キーストアの使用に使用される **io.vertx.ext.auth.KeyStoreOptions**
- 対称シークレットの処理に使用される **io.vertx.ext.auth.SecretOptions**
- 公開鍵の処理に使用される **io.vertx.ext.auth.PubSecKeyOptions**

以下のセクションでは、キー管理の変更点を説明します。

6.3.1. シークレットオプションクラスが利用できない

SecretOptions クラスは利用できなくなりました。新しい **PubSecKeyOptions** クラスを使用して、暗号鍵と連携します。

以下の例は、Eclipse Vert.x 3.x リリースで **SecretOptions** クラスのメソッドが使用される方法を表しています。

```
new SecretOptions()
   .setType("HS256")
   .setSecret("password")
```

以下の例は、Eclipse Vert.x 4 で **PubSecKeyOptions** クラスのメソッドを使用する方法を示しています。

```
new PubSecKeyOptions()
   .setAlgorithm("HS256")
   .setSecretKey("password")
```

6.3.2. 公開鍵管理での更新

Eclipse Vert.x 3.x での公開秘密鍵管理の設定オブジェクトは以下を前提とします。

- 鍵はキーペアとして設定します。
- キーデータは、標準の区切り文字なしで PKCS8 でエンコードされた文字列です。

以下の例は、Eclipse Vert.x 3.x でキーペアを設定する方法を示しています。

```
new PubSecKeyOptions()
  .setPublicKey(
    // remove the PEM boundaries
    pubPemString
      .replaceAll("-----BEGIN PUBLIC KEY----")
      .replaceAll("-----END PUBLIC KEY----"))
  .setSecretKey(
    // remove the PEM boundaries
    secPemString
      .replaceAll("-----BEGIN PUBLIC KEY----")
      .replaceAll("-----END PUBLIC KEY----"));
```

Eclipse Vert.x 4 では、公開鍵と秘密鍵の両方を指定する必要があります。

以下の例は、Eclipse Vert.x 4 でキーペアを設定する方法を示しています。

-

```
PubSecKeyOptions pubKey =
  new PubSecKeyOptions()
    // the buffer is the exact contents of the PEM file and had boundaries included in it
    .setBuffer(pubPemString);

PubSecKeyOptions secKey =
  new PubSecKeyOptions()
    // the buffer is the exact contents of the PEM file and had boundaries included in it
    .setBuffer(secPemString);
```

PubSecKeyOptions を使用して X509 証明書に対応できるようになりました。

```
PubSecKeyOptions x509Certificate =
  new PubSecKeyOptions()
    // the buffer is the exact contents of the PEM file and had boundaries included in it
    .setBuffer(x509PemString);
```

6.3.3. キーストア管理の変更点

Eclipse Vert.x 3.x では、**KeyStoreOptions** はキーストアの形式が **jceks** であると仮定し、保存されたパスワードはキーのパスワードと同じであることを前提としています。**jceks** はプロプライエタリー形式であるため、代わりに JDK などの標準形式を使用することが推奨されます。

Eclipse Vert.x 4 で **KeyStoreOptions** を使用する場合は、ストアタイプを指定できます。たとえば、PKCS11 や PKCS12 などのストアタイプを設定できます。デフォルトのストアタイプは **jceks** です。

Eclipse Vert.x 3.x では、すべてのキーストアエントリーが同じパスワード (キーストアパスワード) を共有します。Eclipse Vert.x 4 では、各キーストアエントリーに専用のパスワードを指定できます。各キーストアエントリーにパスワードを設定しない場合は、キーストアパスワードをすべてのエントリーのデフォルトパスワードとして設定できます。

以下の例は、Eclipse Vert.x 3.x で **jceks** キーストアを読み込む方法を示しています。

```
new KeyStoreOptions()
  .setPath("path/to/keystore.jks")
  .setPassword("keystore-password");
```

Eclipse Vert.x 4 では、デフォルトの形式は JDK によって設定されるデフォルトの形式であることを前提としています。形式は、Java 9 以降の **PKCS12** です。

以下の例は、Eclipse Vert.x 4 で **jceks** キーストアを読み込む方法を示しています。

```
new KeyStoreOptions()
  .setPath("path/to/keystore.jks")
  // Modern JDKs use `jceks` keystore. But this type is not the default
  // If the type is not set to `jceks` then probably `pkcs12` will be used
  .setType("jceks")
  .setPassword("keystore-password")
  // optionally if your keys have different passwords
  // and if a key specific id is not provided it defaults to
  // the keystore password
  .putPasswordProtection("key-id", "key-specific-password");
```

6.4. 非推奨になった、および削除された認証および承認のメソッド

以下のセクションでは、認証用および承認用のメソッドで、非推奨となったもの、および削除されたものの一覧を示します。

6.4.1. 削除された認証メソッドおよび承認メソッドの一覧

以下のメソッドが削除されました。

削除されたメソッド	置き換えメソッド
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()**
User.setAuthProvider()	置き換えるメソッドなし
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
Oauth2ClientOptions.isUseAuthorizationHeader()	置き換えるメソッドなし
Oauth2ClientOptions.scopeSeparator()	置き換えるメソッドなし

6.4.2. 非推奨の認証および承認のメソッドの一覧

以下の方法が非推奨になりました。

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()**

非推奨となったメソッド	置き換えメソッド
Oauth2ClientOptions.isUseAuthorizationHeader()	置き換えるメソッドなし

6.4.3. 非推奨の認証クラスおよび承認クラスの一覧

以下のクラスが非推奨になりました。

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
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

第7章 プロトコルの変更点

本セクションでは、ネットワークプロトコルの変更を説明します。

7.1. ECLIPSE VERT.X GRPC の変更点

以下のセクションでは、Eclipse Vert.x gRP の変更点を説明します。

7.1.1. 新しい gRPC コンパイラープラグイン

Eclipse Vert.x 4 では、**protoc-gen-grpc-java** モジュールは利用できなくなりました。このモジュールは、公式の gRPC コンパイラーのフォークです。以前のリリースの Eclipse Vert.x では、このフォークを使用して作業する必要がありました。このフォークは Eclipse プロジェクトにより維持されます。フォークの使用は複雑です。

以前のリリースでは、gRPC を使用するために、以下の情報が **pom.xml** ファイルに追加されていました。

```
  <!-- Vert.x 3.x -->
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <configuration>
    <protocArtifact>com.google.protobuf:protoc:3.2.0:exe:${os.detected.classifier}</protocArtifact>
    <pluginId>grpc-java</pluginId>
    <!-- NOTE: the gav coordinates point to the 3.x only compiler fork -->
    <pluginArtifact>io.vertx:protoc-gen-grpc-
java:${vertx.grpc.version}:exe:${os.detected.classifier}</pluginArtifact>
  </configuration>
  ...
</plugin>
```

Eclipse Vert.x 4 では、新しい gRPC コンパイラープラグインが利用できるようになりました。このプラグインは、フォークの代わりに公式の gRPC コンパイラーを使用します。新しい gRPC プラグインを使用するには、以下の情報を **pom.xml** ファイルに追加します。

```
  <!-- Vert.x 4.0 -->
<plugin>
  <groupId>org.xolstice.maven.plugins</groupId>
  <artifactId>protobuf-maven-plugin</artifactId>
  <configuration>
    <protocArtifact>com.google.protobuf:protoc:3.2.0:exe:${os.detected.classifier}</protocArtifact>
    <pluginId>grpc-java</pluginId>
    <!-- NOTE: the gav coordinates point to the official compiler -->
    <pluginArtifact>io.grpc:protoc-gen-grpc-
java:${vertx.grpc.version}:exe:${os.detected.classifier}</pluginArtifact>
    <protocPlugins>
      <!-- NEW: a plugin is added to generate the Vert.x specific code -->
      <protocPlugin>
        <id>vertx-grpc-protoc-plugin</id>
        <groupId>io.vertx</groupId>
        <artifactId>vertx-grpc-protoc-plugin</artifactId>
        <version>${vertx.version}</version>
        <mainClass>io.vertx.grpc.protoc.plugin.VertxGrpcGenerator</mainClass>
```

```
      </protocPlugin>
    </protocPlugins>
  </configuration>
  ...
</plugin>
```

7.1.2. 生成されたコードの移行

Eclipse Vert.x 4 では、新しいコンパイラーが使用されます。新しい gRPC プラグインが使用される場合、生成されたコードは同じソースファイルに書き込まれません。これは、コンパイラーがベースクラスでのカスタムコード生成を許可しないためです。コードを保存するには、プラグインで異なる名前を持つ新しいクラスを生成する必要があります。

以前のバージョンの Eclipse Vert.x では、古い gRPC プラグインが同じソースファイルに生成されたコードを書き込みました。

たとえば、以下の記述子があるとします。

```
service Greeter {
  rpc SayHello (HelloRequest) returns (HelloReply) {}
}
```

Eclipse Vert.x 3.x では、コードは **GreeterGrpc** クラスに生成されます。

```
// 3.x
GreeterGrpc.GreeterVertxImplBase service =
  new GreeterGrpc.GreeterVertxImplBase() {
      ...
  }
```

Eclipse Vert.x 4 では、コードは **VertxGreeterGrpc** クラスで生成されます。

```
// 4.x
VertxGreeterGrpc.GreeterVertxImplBase service =
  new VertxGreeterGrpc.GreeterVertxImplBase() {
      ...
  }
```

7.1.3. gRPC API による future のサポート

Eclipse Vert.x 4 では、gRPC API は future をサポートします。gRPC プラグインはプロミスキャス化された API を生成します。これらの API は標準の Eclipse Vert.x の入力引数および出力引数を使用するため、標準の Eclipse Vert.x アプリケーションの作成が容易になります。

以下の例は、Eclipse Vert.x 3.x での promise の使用例を示しています。

```
// 3.x
GreeterGrpc.GreeterVertxImplBase service =
  new GreeterGrpc.GreeterVertxImplBase() {
      @Override
      public void sayHello(HelloRequest request, Promise<HelloReply> future) {
        future.complete(
```

```
            HelloReply.newBuilder().setMessage(request.getName()).build());
      }
  }
```

以下の例は、Eclipse Vert.x 4 での future の使用を示しています。

```
// 4.x
VertxGreeterGrpc.GreeterVertxImplBase service =
  new VertxGreeterGrpc.GreeterVertxImplBase() {
    @Override
    public Future<HelloReply> sayHello(HelloRequest request) {
      return Future.succeededFuture(
        HelloReply.newBuilder()
          .setMessage(request.getName())
          .build());
    }
  }
```

7.2. ECLIPSE VERT.X MQTT の変更点

以下のセクションでは、Eclipse Vert.x MQTT の変更点を説明します。

7.2.1. MQTT クライアントの一部の fluent メソッドが future を返す

MqttClient クラスの fluent メソッドの一部は、fluent ではなく **Future** を返します。たとえば、**MqttClient.connect()**、**MqttClient.disconnect()**、**MqttClient.publish()** などのメソッドは、Eclipse Vert.x 4 の future を返します。

以下の例は、Eclipse Vert.x 3.x リリースでの **publish()** メソッドの使用を示しています。

```
client
  .publish("hello", Buffer.buffer("hello"), MqttQoS.EXACTLY_ONCE, false, false)
  .publish("hello", Buffer.buffer("hello"), MqttQoS.AT_LEAST_ONCE, false, false);
```

以下の例は、Eclipse Vert.x 4 リリースの **publish()** メソッドの使用を示しています。

```
client.publish("hello", Buffer.buffer("hello"), MqttQoS.EXACTLY_ONCE, false, false);
client.publish("hello", Buffer.buffer("hello"), MqttQoS.AT_LEAST_ONCE, false, false);
```

7.2.2. MqttWill メッセージはバッファーを返す

MqttWill データオブジェクトは、文字列メッセージをバイトアレイではなく Eclipse Vert.x バッファーとしてラップします。

7.2.3. MQTT から非推奨の MqttWill および承認メソッドを削除

以下の MQTT メソッドが削除されました。

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()**

7.3. ECLIPSE VERT.X サービスプロキシーの変更点

以下のセクションでは、サービスプロキシーの変更点を説明します。

7.3.1. サービスプロキシーコードジェネレーターの使用

ServiceProxyProcessor クラスが削除されました。

サービスプロキシーコードジェネレーターを使用するには、クラスパスのプロセッサー分類子で **vertx-codegen** をインポートする必要があります。

```
<dependencies>
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-codegen</artifactId>
    <classifier>processor</classifier>
  </dependency>
  <dependency>
    <groupId>io.vertx</groupId>
    <artifactId>vertx-service-proxy</artifactId>
  </dependency>
</dependencies>
```

サービスプロキシーは、**vertx-codegen** から **io.vertx.codegen.CodeGenProcessor** を再利用して、サービスプロキシーおよびハンドラーのコード生成を開始します。

第8章 クライアントコンポーネントの変更

本セクションでは、Eclipse Vert.x クライアントの変更点を説明します。

8.1. ECLIPSE VERT.X KAFKA クライアントの変更点

Eclipse Vert.x Kafka クライアントの変更点を説明します。

8.1.1. AdminUtils クラスが利用できなくなる

AdminUtils クラスが利用できなくなりました。代わりに新しい **KafkaAdminClient** クラスを使用して、Kafka クラスターで管理操作を実行します。

8.1.2. フラッシュメソッドが非同期ハンドラーを使用

KafkaProducer クラスのフラッシュメソッドは、**Handler<Void>** の代わりに **Handler<AsyncResult<Void>>** を使用します。

8.2. ECLIPSE VERT.X JDBC クライアントの変更点

Eclipse Vert.x 4 以降、JDBC クライアントは SQL クライアントをサポートします。SQL 共通モジュールは、**io.vertx:vertx-jdbc-client** モジュールにマージされる **io.vertx:vertx-sql-common** という JDBC クライアントでマージされました。**io.vertx:vertx-jdbc-client** にはそれが含まれるため、**io.vertx:vertx-sql-common** 依存関係ファイルを削除する必要があります。SQL 共通クライアントのマージにより、すべてのデータベース API が JDBC クライアントに統合されました。

Eclipse Vert.x 4 では、SQL クライアントが更新され、以下のクライアントが含まれるようになりました。

- リアクティブ PostgreSQL クライアント。以前のリリースでは、リアクティブ PostgreSQL クライアントが含まれていました。
- リアクティブ MySQL クライアント
- リアクティブ DB2 クライアント
- リアクティブ PostgreSQL クライアントを引き続き組み込みます。このクライアントは Eclipse Vert.x 3.x リリースでも利用できていました。
- 既存の JDBC クライアントに JDBC クライアント API と SQL クライアント API の両方が含まれるようになりました。

リアクティブ実装では、データベースネットワークプロトコルを使用します。これにより、リソース効率が向上します。

データベースへの JDBC 呼び出しはブロック呼び出しです。JDBC クライアントはワーカースレッドを使用してこれらの呼び出しをブロックしません。

以下のセクションでは、Eclipse Vert.x JDBC クライアントの変更点を説明します。

8.2.1. プールの作成

Eclipse Vert.x 4 では、JDBC クライアント API を使用してプールを作成できます。以前のリリースでは、クライアントのみを作成できました。プールを作成できませんでした。

以下の例は、Eclipse Vert.x 3.x でクライアントを作成する方法を示しています。

```
// 3.x
SQLClient client = JDBCClient.create(vertx, jsonConfig);
```

以下の例は、Eclipse Vert.x 4 でプールを作成する方法を示しています。

```
// 4.x
JDBCPool pool = JDBCPool.pool(vertx, jsonConfig);
```

注記

Eclipse Vert.x 3.x API は Eclipse Vert.x 4 でサポートされていますが、アプリケーションで新しい JDBC クライアント API を使用することが推奨されます。

プールを使用すると、単純なクエリーを実行できます。単純なクエリーの接続を管理する必要はありません。ただし、複雑なクエリーや複数のクエリーの場合は、接続を管理する必要があります。

以下の例は、Eclipse Vert.x 3.x でクエリーの接続を管理する方法を示しています。

```
// 3.x
client.getConnection(res -> {
  if (res.succeeded()) {
    SQLConnection connection = res.result();
    // Important, do not forget to return the connection
    connection.close();
  } else {
    // Failed to get connection
  }
});
```

以下の例は、Eclipse Vert.x 4 でクエリーの接続を管理する方法を示しています。

```
// 4.x
pool
  .getConnection()
  .onFailure(e -> {
    // Failed to get a connection
  })
  .onSuccess(conn -> {
    // Important, do not forget to return the connection
    conn.close();
  });
```

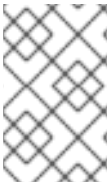
8.2.2. Typsesafe 設定のサポート

設定には **jsonConfig** を使用できます。ただし、**jsonConfig** を使用するとエラーが発生する可能性があります。これらのエラーを回避するために、JDBC クライアントでは Typesafe Config が導入されました。

以下の例は、Typesafe Config の基本的な構造を示しています。

```
// 4.x ONLY!!!
```

```
JDBCPool pool = JDBCPool.pool(
    vertx,
    // configure the connection
    new JDBCConnectOptions()
      // H2 connection string
      .setJdbcUrl("jdbc:h2:~/test")
      // username
      .setUser("sa")
      // password
      .setPassword(""),
    // configure the pool
    new PoolOptions()
      .setMaxSize(16)
  );
```



注記

Typesafe Config を使用するには、プロジェクトに **agroal** 接続プールを含める必要があります。プールは多くの設定オプションを公開しないため、設定を簡単に使用できるようになります。

8.2.3. SQL クエリーの実行

ここでは、JDBC クライアントでクエリーを実行する方法を説明します。

8.2.3.1. 単発のクエリーの実行

以下の例は、Eclipse Vert.x 3.x で接続を管理せずにクエリーを実行する方法を示しています。

```
// 3.x
client.query("SELECT * FROM user WHERE emp_id > ?", new JsonArray().add(1000), res -> {
  if (res.succeeded()) {
    ResultSet rs = res2.result();
    // You can use these results in your application
  }
});
```

以下の例は、Eclipse Vert.x 4 で接続を管理せずにクエリーを実行する方法を示しています。

```
// 4.x
pool
  .preparedQuery("SELECT * FROM user WHERE emp_id > ?")
  // the emp id to look up
  .execute(Tuple.of(1000))
  .onSuccess(rows -> {
    for (Row row : rows) {
      System.out.println(row.getString("FIRST_NAME"));
    }
  });
```

8.2.3.2. 管理接続でのクエリーの実行

以下の例は、Eclipse Vert.x 4 で管理された接続でクエリーを実行する方法を示しています。

-

```
pool
  .getConnection()
  .onFailure(e -> {
    // Failed to get a connection
  })
  .onSuccess(conn -> {
    conn
      .query("SELECT * FROM user")
      .execute()
      .onFailure(e -> {
        // Handle the failure
        // Important, do not forget to return the connection
        conn.close();
      })
      .onSuccess(rows -> {
        for (Row row : rows) {
          System.out.println(row.getString("FIRST_NAME"));
        }
        // Important, do not forget to return the connection
        conn.close();
      });
  });
```

8.2.4. ストアドプロシージャーのサポート

ストアドプロシージャーは JDBC クライアントでサポートされます。

以下の例は、Eclipse Vert.x 3.x で **IN** 引数を渡す方法を示しています。

```
// 3.x
connection.callWithParams(
  "{ call new_customer(?, ?) }",
  new JsonArray().add("John").add("Doe"),
  null,
  res -> {
    if (res.succeeded()) {
      // Success!
    } else {
      // Failed!
    }
  });
```

以下の例は、Eclipse Vert.x 4 で **IN** 引数を渡す方法を示しています。

```
// 4.0
client
  .preparedQuery("{call new_customer(?, ?)}")
  .execute(Tuple.of("Paulo", "Lopes"))
  .onSuccess(rows -> {
    ...
  });
```

Eclipse Vert.x 3.x では、**IN** 引数および **OUT** 引数を組み合わせるサポートは、利用可能なタイプにより大幅に制限されていました。Eclipse Vert.x 4 では、プールは安全なタイプであり、**IN** と **OUT** の引数の組み合わせを処理できます。アプリケーションで **INOUT** パラメーターを使用することもできます。

以下の例は、Eclipse Vert.x 3.x での引数の処理を示しています。

```
// 3.x
connection.callWithParams(
  "{ call customer_lastname(?, ?) }",
  new JsonArray().add("John"),
  new JsonArray().addNull().add("VARCHAR"),
  res -> {
    if (res.succeeded()) {
      ResultSet result = res.result();
    } else {
      // Failed!
    }
});
```

以下の例は、Eclipse Vert.x 4 での引数の処理を示しています。

```
// 4.x
client
  .preparedQuery("{call customer_lastname(?, ?)}")
  .execute(Tuple.of("John", SqlOutParam.OUT(JDBCType.VARCHAR)))
  .onSuccess(rows -> {
    ...
  });
```

JDBC クライアントで、データ型が更新されました。

- **OUT** 型の引数に、戻り値のタイプを指定できます。この例では、**OUT** 引数は、JDBC 定数である **VARCHAR** 型として指定されています。
- タイプは JSON の制限によってバインドされません。タイプ名のテキスト定数の代わりに、データベース固有の型を使用できるようになりました。

8.3. ECLIPSE VERT.X メールクライアントの変更点

以下のセクションでは、Eclipse Vert.x メールクライアントの変更点を説明します。

8.3.1. MailAttachment がインターフェースとして利用可能

Eclipse Vert.x 4 以降では、**MailAttachment** はインターフェースとして利用できます。これにより、ストリームでメール添付機能を使用できます。以前のリリースの Eclipse Vert.x では、**MailAttachment** はクラスとして使用可能であり、メールの添付ファイルはデータオブジェクトとして表されていました。

8.3.2. メール設定インターフェースが net クライアントオプションを拡張

MailConfig インターフェースは **NetClientOptions** インターフェースを拡張します。この拡張機能により、メール設定は **NetClient** のプロキシー設定もサポートします。

8.4. ECLIPSE VERT.X AMQP クライアントの変更点

Eclipse Vert.x AMQP クライアントの変更点を説明します。

8.4.1. AmqpMessage 引数が含まれる AMQP クライアントのメソッドを削除

Handler<AmqpMessage> を引数として持つ AMQP クライアントメソッドが削除されました。以前のリリースでは、このハンドラーを **ReadStream<AmqpMessage>** に設定できました。ただし、アプリケーションを移行して future を使用する場合、このような方法は使用できません。

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)**

8.5. ECLIPSE VERT.X MONGODB クライアントの変更点

以下のセクションでは、Eclipse Vert.x MongoDB クライアントの変更点を説明します。

8.5.1. MongoDB クライアントから削除されたメソッド

以下のメソッドは **MongoClient** クラスから削除されました。

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()**
MongoClient.removeWithOptions()	**MongoClient.removeDocumentsWithOptions()**
MongoClient.removeOne()	**MongoClient.removeDocument()**

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

8.6. EVENTBUS JAVASCRIPT クライアントの変更点

Eclipse Vert.x 4 では、EventBus JavaScript クライアントモジュールが新しい場所で利用できます。新しい場所からモジュールを使用するようにビルドシステムを更新する必要があります。

Eclipse Vert.x 3.x では、イベントバスの JavaScript クライアントはさまざまな場所で利用できました。以下に例を示します。

- Maven Central
- NPM
- Bower.io
- CDNJS
- webjars

Eclipse Vert.x 4 では、JavaScript クライアントは **npm** でのみ利用できます。EventBus JavaScript クライアントモジュールは、以下の場所からアクセスできます。

- CDN
- npm パッケージ
  ビルドスクリプトで以下のコードを使用してモジュールにアクセスします。
  - JSON スクリプト

```
{
  "devDependencies": {
    "@vertx/eventbus-bridge-client.js": "1.0.0-1"
  }
}
```

  - XML スクリプト

```
<dependency>
    <groupId>org.webjars.npm</groupId>
    <artifactId>vertx__eventbus-bridge-client.js</artifactId>
    <version>1.0.0-1</version>
</dependency>
```

8.6.1. JavaScript クライアントのバージョン管理

Eclipse Vert.x 4 より前は、すべての Eclipse Vert.x リリースに JavaScript クライアントの新しいリリースが含まれていました。

しかし、Eclipse Vert.x 4 以降では、クライアントに変更がある場合に限り、新しいバージョンの JavaScript クライアントを npm で利用できます。バージョン変更がなければ、すべての Eclipse Vert.x リリースでクライアントアプリケーションを更新する必要はありません。

8.7. ECLIPSE VERT.X REDIS クライアントの変更点

Eclipse Vert.x 4 では、**Redis** クラスを使用して Redis クライアントと連携します。**RedisClient** クラスは利用できなくなりました。

注記

アプリケーションを **RedisClient** クラスから **Redis** クラスに移行しやすくするため、ヘルパークラス **RedisAPI** が利用できるようになりました。**RedisAPI** により、**RedisClient** クラスと同様の機能を複製できます。

新しいクラスには、プロトコルおよび Redis サーバー機能の機能強化がすべて含まれます。新規クラスを使用して以下を行います。

- すべての Redis コマンドとの連携
- 単一サーバーに接続
- Redis Sentinel が有効になっている高可用性サーバーに接続
- Redis のクラスター設定への接続
- Redis 拡張機能での要求の実行
- RESP2 および RESP3 サーバープロトコルサーバーの両方と通信

8.7.1. 既存の Redis クライアントから新規クライアントへの移行

既存のアプリケーションを新しい **Redis** クライアントに直接移行するか、またはヘルパークラス **RedisAPI** を使用して 2 つの手順でアプリケーションを移行できます。

アプリケーションを移行する前に、クライアントを作成する必要があります。

8.7.1.1. クライアントの作成

以下の例は、Eclipse Vert.x 3.x リリースでの Redis クライアントの作成方法を示しています。

```
// Create the redis client (3.x)
RedisClient client = RedisClient
  .create(vertx, new RedisOptions().setHost(host));
```

以下の例は、Eclipse Vert.x 4 で Redis クライアントを作成する方法を示しています。

```
// Create the redis client (4.x)
Redis client = Redis
  .createClient(
    vertx,
    "redis://server.address:port");
```

Eclipse Vert.x 4 では、クライアントは以下の標準の接続文字列構文を使用します。

```
redis[s]://[[user]:password@]server[:port]/[database]
```

8.7.1.2. アプリケーションの RedisAPI への移行

「RedisAPI」を使用して、接続の管理方法を決定できるようになりました。

- クライアントがプールを使用する接続を管理できるようにします。

または、以下を実行します。

- 新しい接続を要求すると、接続を制御できます。完了したら必ず接続を閉じるか、または返す必要があります。

クライアントを作成してから、要求を処理するために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 3.x リリースでクライアントを作成した後に要求を処理する方法を示しています。

```
// Using 3.x
// omitting the error handling for brevity
client.set("key", "value", s -> {
  if (s.succeeded()) {
    System.out.println("key stored");
    client.get("key", g -> {
      if (s.succeeded()) {
        System.out.println("Retrieved value: " + s.result());
      }
    });
  }
});
```

以下の例は、Eclipse Vert.x 4 でクライアントを作成した後に要求を処理する方法を示しています。この例では、ハードコーディングオプションの代わりに、キーと値のペアを設定するリストを使用します。コマンドで使用できる引数の詳細は、「Redis SET コマンド」を参照してください。

```
// Using 4.x
// omitting the error handling for brevity

// 1. Wrap the client into a RedisAPI
api = RedisAPI.api(client);

// 2. Use the typed API
api.set(
    Arrays.asList("key", "value"), s -> {
    if (s.succeeded()) {
      System.out.println("key stored");
      client.get("key", g -> {
        if (s.succeeded()) {
          System.out.println("Retrieved value: " + s.result());
        }
      });
    }
});
```

8.7.1.3. アプリケーションを Redis クライアントに直接移行

新しい **Redis** クライアントに直接移行する場合:

- すべての新しい Redis コマンドを使用できる。
- 拡張機能を使用できる。
- ヘルパークラスから新しいクライアントへの変換が一部減る場合があり、アプリケーションのパフォーマンスが向上する可能性がある。

クライアントを作成してから、要求を処理するために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 3.x リリースでクライアントを作成した後に要求を設定および取得する方法を示しています。

```
// Using 3.x
// omitting the error handling for brevity
client.set("key", "value", s -> {
  if (s.succeeded()) {
    System.out.println("key stored");
    client.get("key", g -> {
      if (s.succeeded()) {
        System.out.println("Retrieved value: " + s.result());
      }
    });
  }
});
```

以下の例は、Eclipse Vert.x 4 でクライアントを作成した後に要求を処理する方法を示しています。

```
// Using 4.x
// omitting the error handling for brevity

import static io.vertx.redis.client.Request.cmd;
import static io.vertx.redis.client.Command.*;

client.send(cmd(SET).arg("key").arg("value"), s -> {
    if (s.succeeded()) {
      System.out.println("key stored");
      client.send(cmd(GET).arg("key"), g -> {
        if (s.succeeded()) {
          System.out.println("Retrieved value: " + s.result());
        }
      });
    }
});
```

Eclipse Vert.x 4 では、すべての対話は **send(Request)** メソッドを使用します。

8.7.1.4. 応答の移行

Eclipse Vert.x 3.x では、クライアントは、Redis 5 までの既知のコマンドをすべてハードコードするのに使用されると、応答もコマンドに従って入力されました。

新しいクライアントでは、コマンドはハードコーディングされません。応答のタイプは **Response** です。新しい Wire プロトコルにはさらに多くのタイプがあります。

古いクライアントでは、応答は以下のようになります。

- **null**
- **Long**
- **String**
- **JsonArray**
- **JsonObject** (アレイ応答 **INFO** および **HMGET** の場合)

新しいクライアントでは、レスポンスは以下のタイプになります。

- **null**
- **Response**

Response オブジェクトには型コンバーターがあります。たとえば、以下のようなコンバーターがあります。

- **toString()**
- **toInteger()**
- **toBoolean()**
- **toBuffer()**

受信したデータが要求されたタイプでない場合、型コンバーターはこれを可能な限り最も近いデータ型に変換します。特定の型への変換ができない場合は、**UnsupportedOperationException** が出力されます。たとえば、**String** から **List** または **Map** への変換はできません。

Response オブジェクトは **Iterable** インターフェースを実装するため、コレクションを処理することもできます。

以下の例は、MGET 要求の実行方法を示しています。

```
// Using 4.x
// omitting the error handling for brevity

import static io.vertx.redis.client.Request.cmd;
import static io.vertx.redis.client.Command.*;

client.send(cmd(MGET).arg("key1").arg("key2").arg("key3"), mget -> {
  mget.result()
    .forEach(value -> {
      // Use the single value
```

8.7.2. Eclipse Vert.x Redis クライアントの更新

本セクションでは、Redis クライアントの変更点を説明します。

8.7.2.1. Redis ロールおよびノードのオプションから非推奨の用語「slave (スレーブ)」を削除

Redis ロールおよびノードオプションで使用されていた非推奨の「slave (スレーブ)」という用語が、「replica」に置き換えられました。

ロール

以下の例は、Eclipse Vert.x 3.x リリースでの **SLAVE** ロールの使用を示しています。

```
// Before (3.x)
Redis.createClient(
    rule.vertx(),
    new RedisOptions()
      .setType(RedisClientType.SENTINEL)
      .addConnectionString("redis://localhost:5000")
      .setMasterName("sentinel7000")
      .setRole(RedisRole.SLAVE));
```

以下の例は、Eclipse Vert.x 4 での **REPLICA** ロールの使用を示しています。

```
// After (4.0)
Redis.createClient(
    rule.vertx(),
    new RedisOptions()
      .setType(RedisClientType.SENTINEL)
      .addConnectionString("redis://localhost:5000")
      .setMasterName("sentinel7000")
      .setRole(RedisRole.REPLICA));
```

ノードのオプション

以下の例は、Eclipse Vert.x 3.x リリースでのノード型の **RedisSlaves** の使用方法を示しています。

```
// Before (3.9)
options.setUseSlaves(RedisSlaves);
```

以下の例は、Eclipse Vert.x 4 でのノード型の **RedisReplicas** の使用方法を示しています。

```
// After (4.0)
options.setUseReplicas(RedisReplicas);
```

第9章 クラスタリングの変更点

本セクションでは、クラスタリングの変更点を説明します。

9.1. クラスター化されたフラグがオプションクラスから削除

Eclipse Vert.x アプリケーションでクラスタリングを指定、取得、および設定するのに使用されるメソッドおよびブール値が **VertxOptions** クラスおよび **EventBusOptions** クラスから削除されました。

9.2. INFINISPAN クラスターマネージャーの変更点

ここでは、Infinispan クラスターマネージャーの変更点を説明します。

9.2.1. カスタム設定の更新

Infinispan クラスターマネージャーは Infinispan 11 をベースとしています。

Eclipse Vert.x 4 では、クラスタリング SPI が再設計されました。サブスクリプションのデータモデルが変更されました。そのため、Eclipse Vert.x 3.x ノードと Eclipse Vert.x 4 ノードを同じ Infinispan クラスターに追加することはできません。

EventBus API および SharedData API が同じままであるため、Eclipse Vert.x アプリケーションはこの変更の影響を受けません。

Eclipse Vert.x 3.x アプリケーションでカスタム Infinispan 設定ファイルがある場合:

- **__vertx.subs** キャッシュタイプを Distributed ではなく Replicated に変更します。
- レプリケートされた **__vertx.nodeInfo** を追加します。

```
<cache-container default-cache="distributed-cache">
  <distributed-cache name="distributed-cache"/>
  <replicated-cache name="__vertx.subs"/>
  <replicated-cache name="__vertx.haInfo"/>
  <replicated-cache name="__vertx.nodeInfo"/>
  <distributed-cache-configuration name="__vertx.distributed.cache.configuration"/>
 </cache-container>
```

Openshift で Eclipse Vert.x クラスターを実行する場合は、**infinispan-cloud** JAR が不要になりました。JAR はビルドファイルの dependencies セクションから削除されています。この JAR にバンドルされた設定ファイルは **infinispan-core** JAR に含まれるようになりました。

9.3. クラスターの移行

コードベースの移行ストラテジーを決定することが重要です。これは、以下の理由により、Eclipse Vert.x 3.x ノードと Eclipse Vert.x 4 ノードを単一のクラスターに追加できないためです。

- クラスターマネージャーのアップグレード - クラスターマネージャーのメジャーバージョンのアップグレードにより、後方互換性が回避されます。
- サブスクリプションデータの変更点 - Eclipse Vert.x は、クラスターマネージャーに保存されている EventBus サブスクリプションデータの形式を変更しました。

- トランスポートプロトコルの変更点 - Eclipse Vert.x がクラスターのメッセージトランスポートプロトコルのフィールドを変更しました。

単一アプリケーションまたは密接に関連するマイクロサービスに Eclipse Vert.x クラスターがある場合は、コードベースを一度に新規クラスターに移行できます。

ただし、一度にコードベースを移行できない場合は、本セクションの推奨事項を使用して Eclipse Vert.x 3.x コードベースを Eclipse Vert.x 4 に移行します。

9.3.1. クラスターの分割

アプリケーションごとに異なるチームが verticle をデプロイしたクラスターがある場合は、Eclipse Vert.x 3.x クラスターを小規模なものに分割することを検討してください。クラスターを分割すると、分離されたコンポーネントはクラスタリング機能を使用して通信できなくなることに注意してください。以下のコンポーネントを使用してクラスターを分割できます。

- EventBus 要求および応答 - HTTP または RESTful Web サービス、gRPC
- EventBus 送信および公開 - メッセージングシステム、Postgres **LISTEN** および **NOTIFY**、Redis Pub および Sub
- 共有データ - Redis、Infinispan

クラスターの分割後に、各チームの準備ができたら、または必要に応じて Eclipse Vert.x 4 に移行できます。

9.3.2. Eclipse Vert.x EventBus リンクの使用

クラスターを分割できない場合は、Vert.x EventBus リンク を使用してコードベースを段階的に移行します。

Vert.x EventBus Link は、Eclipse Vert.x 3.x クラスター EventBus を Eclipse Vert.x 4 クラスター EventBus に接続するツールです。

警告

共有データ API、つまりマップ、カウンター、およびロックの移行はサポートされません。

このツールは、**EventBus** インターフェースを実装する **EventBusLink** オブジェクトを作成します。**EventBusLink** のインスタンスは、各クラスターの少なくとも 1 つのノードに作成されます。インスタンスは、アドレスのセットを提供し、その動作はメッセージのパラダイムによって異なります。

- **fire** および **forget** ならびに **request** および **reply**: メッセージはリモートクラスターに送信されます。
- **publish**: メッセージは、このクラスターとリモートクラスターの両方に送信されます。

Eclipse Vert.x EventBus Link リンクは、WebSocket サーバーを作成してメッセージを受け取り、WebSocket クライアントを使用してメッセージを送信します。

詳細は、「getting started」および「using」を参照してください。

第10章 ECLIPSE VERT.X のその他の変更点

以下のセクションでは、Eclipse Vert.x 4 のその他の変更点を説明します。

10.1. STARTER クラスの削除

Starter クラスは削除されました。代わりに **Launcher** クラスを使用して、**main()** メソッドなしで Eclipse Vert.x アプリケーションを起動します。

10.2. JAVA 8 の分離デプロイメント

Eclipse Vert.x 4 は Java 11 をサポートします。この Java バージョンでは、分離されたクラスローディングはサポートされません。Eclipse Vert.x 4 では、Java 8 で分離されたクラスローディングがサポートされます。

10.3. ECLIPSE VERT.X コンテキストから HOOK メソッドを削除

Context.addCloseHook() メソッドおよび **Context.removeCloseHook()** メソッドは **Context** クラスから削除されました。これらのメソッドは内部インターフェース **InternalContext** に移動しました。

10.4. オプションから CLONE メソッドを削除

KeyCertOptions.clone() メソッド、**TrustOptions.clone()** メソッド、および **SSLEngineOptions.clone()** メソッドが削除されました。代わりに **KeyCertOptions.copy()** メソッド、**TrustOptions.copy()** メソッド、および **SSLEngineOptions.copy()** メソッドを使用してください。

10.5. オプションから EQUALS メソッドおよび HASHCODE メソッドが削除

VertxOptions.equals() メソッドおよび **VertxOptions.hashCode()** メソッドが削除されました。

10.6. ファイルのキャッシュを確認する新しいメソッド

VertxOptions.fileResolverCachingEnabled() メソッドが削除されました。代わりに、**FileSystemOptions.isFileCachingEnabled()** メソッドを使用して、クラスパスを解決するのにファイルキャッシュが有効かどうかを確認します。

10.7. SERVICE PROVIDER INTERFACE (SPI) メトリクス

Metrics.isEnabled() メソッドが削除されました。Service Provider Interface (SPI) メトリクスは null オブジェクトを返し、メトリクスが有効ではないことを示します。

10.8. プールされたバッファーメソッドの削除

プールされたバッファーメソッド **TCPSSLOptions.isUsePooledBuffers()** および **TCPSSLOptions.setUsePooledBuffers()** が削除されました。

10.9. 共有データソースのないクライアントを作成するメソッド

次の新しいメソッドを使用して、他のクライアントとデータソースを共有していないクライアントを作成します。これらのメソッドは、独自のデータソースを維持します。

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()**

10.10. ECLIPSE VERT.X JUNIT5 の変更点

以下のセクションでは、Eclipse Vert.x JUnit5 の変更点を説明します。

10.10.1. vertx-core モジュールと、拡張機能の更新をサポート

vertx-core モジュールが更新され、パラメーターの挿入にサービスプロバイダーインターフェースが使用されるようになりました。この変更により、JUnit5 に以下の更新が行われました。

- 作成に必要なパラメーターの前に **Vertx** パラメーターを呼び出す必要があります。たとえば、**WebClient** を挿入する場合などです。
- **vertx-junit5** モジュールは、**vertx-core** モジュールのみをサポートします。
- **reactiverse-junit5-extensions** モジュールは、**WebClient** などの追加パラメータータイプが含まれる拡張機能をホストします。
- RxJava 1 および 2 のバインディングが、**vertx-junit5-extensions** リポジトリーの **vertx-junit5-rx-java** モジュールおよび **vertx-junit5-rx-java2** モジュールとして利用できるようになりました。

10.10.2. Eclipse Vert.x テキストコンテキストで非推奨になった succeeding メソッドおよび failing メソッド

VertxTestContext.succeeding() メソッドおよび **VertxTestContext.failing()** メソッドが非推奨になりました。代わりに **VertxTestContext.succeedingThenComplete()** および **VertxTestContext.failingThenComplete()** メソッドを使用してください。