Red Hat build of Eclipse Vert.x 4.3

Eclipse Vert.x 4.3 迁移指南

用于 Eclipse Vert.x 4.3

Last Updated: 2023-04-07

Red Hat build of Eclipse Vert.x 4.3 Eclipse Vert.x 4.3 迁移指南

用于 Eclipse Vert.x 4.3

法律通告

摘要

本指南提供有关如何将 Eclipse Vert.x 3.x 应用程序升级到 Eclipse Vert.x 4。

目录

前言

本指南描述了 Eclipse Vert.x 4 的更新。使用信息将 Eclipse Vert.x 3.x 应用程序升级到 Eclipse Vert.x 4。它提供有关本发行版本中新的、已弃用和不受支持的功能的信息。

根据应用程序中使用的模块，您可以阅读相关部分来了解更多有关 Eclipse Vert.x 4 中的更改的信息。

对红帽文档提供反馈

我们感谢您对我们文档的反馈。要提供反馈，您可以突出显示文档中的文本并添加注释。

本节介绍如何提交反馈。

前提条件

- 登录到红帽客户门户网站。
- 在红帽客户门户网站中，以 多页 HTML 格式查看文档。

流程

要提供反馈，请执行以下步骤：

1. 点击文档右上角的 **反馈** 按钮查看现有反馈。

注意

反馈功能仅在多页 HTML 格式中启用。

2. 高亮标记您要提供反馈的文档中的部分。
3. 点击在高亮文本旁边弹出的 **添加反馈**。
   文本框会出现在页面右侧的 feedback部分中。
4. 在文本框中输入您的反馈，然后点 **Submit**。
   已创建一个文档问题。
5. 要查看问题，请单击反馈视图中的问题跟踪器链接。

使开源包含更多

红帽致力于替换我们的代码、文档和 Web 属性中存在问题的语言。我们从这四个术语开始：master、slave、黑名单和白名单。由于此项工作十分艰巨，这些更改将在即将推出的几个发行版本中逐步实施。有关更多详情，请参阅我们的首席技术官 Chris Wright 提供的消息。

第 1 章 将应用程序配置为使用 ECLIPSE VERT.X

当您开始将应用程序配置为使用 Eclipse Vert.x 时，您必须在应用程序的根目录下的 **pom.x BOM (Bill of Materials)构件中引用 Eclipse Vert.** x BOM (Bill of Materials)构件。BOM 用于设置工件的正确版本。

前提条件

- 基于 Maven 的应用程序

流程

1. 打开 **pom.xml** 文件，将 **io.vertx:vertx-dependencies** 构件添加到 < **dependencyManagement>** 部分。将 类**型指定**为 **pom**，**范围** 指定为 **导入**。

```
<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. 包含以下属性来跟踪 Eclipse Vert.x 和 Eclipse Vert.x Maven 插件的版本。
属性可用于设置在每个版本中更改的值。例如，产品或插件的版本。

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

3. 指定 **vertx-maven-plugin** 作为用于打包应用程序的插件：

```
<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. 包含 **软件**仓库 **和插件，** 以指定包含工件和插件的存储库以构建应用程序：

```
<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>
```

其他资源

- 有关打包 Eclipse Vert.x 应用程序的更多信息，请参阅 Vert.x Maven 插件 文档。

第 2 章 关于 ECLIPSE VERT.X

Eclipse Vert.x 是用于创建被动、非阻塞和在 Java 虚拟机上运行的异步应用程序的工具包。(JVM)。它包含多个组件，可帮助您创建被动应用程序。它设计为云原生。

由于 Eclipse Vert.x 支持异步应用程序，它可用于创建含有大量消息、大型事件处理、HTTP 交互等的应用程序。

第 3 章 ECLIPSE VERT.X 4 中的变化

本节介绍 Eclipse Vert.x 4 和 3.x 版本之间的基本区别。

3.1. 对异步操作使用将来的方法

Eclipse Vert.x 4 使用将来的异步操作。每个回调方法都有对应的未来方法。将来可用于编写异步操作。您可以使用回调和将来的方法组合将基于回调的应用程序迁移到 Eclipse Vert.x 4。但是，您还可以继续使用回调进行异步操作。

以下示例演示了如何将回调用于 Eclipse Vert.x 3.x 版本的异步操作。

```
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();
  }
});
```

以下示例演示了如何将回调和将来的方法一起用于 Eclipse Vert.x 4 中的异步操作。

```
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();
  }
});
```

错误处理对于未来的更好是更佳的。在回调中，您必须在组成的每个阶段处理故障，而在将来您可以在结束时处理故障。在基本应用程序中，您可能不会注意到使用回调和将来的不同。

以下示例演示了如何使用回调来编写两个异步操作。您可以看到错误在每个位置进行处理。

```
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();
  }
});
```

以下示例演示了如何使用回调和将来在 Eclipse Vert.x 4 中编写两个异步操作。这个错误仅在末尾处理一次。

```
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. 不依赖于 JACKSON DATABIND 库

Eclipse Vert.x 中的 JSON 功能依赖于 Jackson 库。jackson Databind 库启用 JSON 的对象映射。

在 Eclipse Vert.x 4 中，Jackson Databind 是一个可选 Maven 依赖项。如果要使用这个依赖项，您必须在 classpath 中明确添加它。

- 如果您要对象映射 JSON，则必须在 **com.fasterxml.jackson.core:jackson-databind** jar 中明确添加项目描述符中的依赖关系。

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

  未来，如果您决定不使用 JSON 的对象映射，您可以删除这个依赖项。

- 如果没有对象映射 JSON，则不需要 Jackson Databind 库。您可以在没有此 jar 的情况下运行您的应用程序。

3.3. 处理弃用和删除

在 Eclipse Vert.x 4 中弃用或删除一些功能和功能。在将应用程序迁移到 Eclipse Vert.x 4 之前，请检查 弃用和删除。

- 一些 API 在 Eclipse Vert.x 3.x 版本中已弃用，新的等效的 API 在该版本中提供。
- 弃用的 API 已在 Eclipse Vert.x 4 中删除。

如果您的应用程序使用已弃用的 API，您应该更新应用程序以使用新的 API。这有助于将应用程序迁移到最新版本的产品。

使用已弃用的 API 时，Java 编译器会生成警告。在将应用迁移到 Eclipse Vert.x 4 时，您可以使用编译器检查已弃用的方法。

以下示例显示了 Eclipse Vert.x 3.x 版本中已弃用的 EventBus 方法。

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

方法 **send (String,String,Handler<AsyncResult<Message>)** 被替换为 Eclipse Vert.x 4 中的方法 **请求 (String,String,Handler<AsyncResult<Message>>**)。

以下示例演示了如何更新应用程序以使用新的方法。

```
// 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 常见组件的更改

本节介绍基本 Eclipse Vert.x 组件中的更改。

4.1. 消息传递的更改

本节介绍消息传递方法中的更改。

4.1.1. 写流中的写入和结束方法不再是模糊的

WriteStream<T>.write （） 和 **WriteStream<T>.end （）** 方法不再是 fluent。

- 写和最终回调方法返回 **void**。
- 其他写入和最终方法返回 **Future<Void>**。

这是有问题的更改。如果您已将虚线用于写入流，请更新您的应用程序。

4.1.2. MessageProducer 不会扩展 WriteStream

MessageProducer 接口不会扩展 **WriteStream** 接口。

在之前的 Eclipse Vert.x 版本中，**MessageProducer** 接口扩展了 **WriteStream** 接口。**MessageProducer** 界面提供对消息后端的支持。泄漏泄漏将减少消息制作者中的学分。如果这些泄漏使用了所有学分，则不会发送消息。

但是，**MessageConsumer** 将继续扩展 **ReadStream**。当 **MessageConsumer** 暂停并且待处理的消息队列已满时，消息将被丢弃。这会继续与 Rx 生成器集成，以构建使用管道的消息。

4.1.3. 从 MessageProducer中删除了发送方法

MessageProducer 界面中的发送方法已被删除。

使用 **MessageProducer<T>.write (T)** 而不是 **MessageProducer<T>.send (T)** 和 **EventBus.request (String,Object,Handler)** 的方法，而不是 **MessageProducer.send (T,Handler)**。

4.2. EVENTBUS 中的变化

以下部分描述了 EventBus 中的更改。

4.2.1. 删除了 EventBus 中的请求响应发送方法

EventBus.send (..., Handler<AsyncResult<Message<T>>) 和 **Message.reply (..., Handler<AsyncResult<Message<T>>)** 方法已被删除。这些方法可能会导致 Eclipse Vert.x 4 中出现过载问题。该方法的版本返回 **Future<Message<T** > 将与触发和忘记版本冲突。

请求响应的消息传递模式应使用新的 **请求和 replyAndRequest** 方法。

- 使用方法 **EventBus.request (..., Handler<AsyncResult<Message<T>>)** 而不是 **EventBus.send (..., Handler<AsyncResult<Message<T>>)** 来发送消息。
- 使用方法 **Message.replyAndRequest (..., Handler<AsyncResult<Message<T>>)** 而不是 **Message.reply (..., Handler<AsyncResult<Message<T>>)** 来回复消息。

以下示例显示了请求并回复 Eclipse Vert.x 3.x 版本中的消息。

Request（请求）

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

答复

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

以下示例显示了请求并回复 Eclipse Vert.x 4 中的消息。

Request（请求）

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

答复

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

4.3. 将来的更改

本节解释了未来的更改。

4.3.1. 为将来的多个处理程序提供支持

从 Eclipse Vert.x 4 开始，未来会支持多个处理程序。**Future<T>.setHandler （）** 方法用于设置单一处理程序并已被删除。使用 **Future<T>.onComplete （）、Future<T> .onSuccess （） 和 Future<T>.onFailure （）** 方法，而是在完成、成功和失败结果上调用处理程序。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中调用处理器。

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

以下示例演示了如何在 Eclipse Vert.x 4 中调用新的 **Future<T>.onComplete （）** 方法。

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

4.3.2. 以后删除了 completer （） 方法

在较早版本的 Eclipse Vert.x 中，您可以使用 **Future.completer （）** 方法访问 **Handler<AsyncResult<T>>**，这与 **Future** 相关联。

在 Eclipse Vert.x 4 中，**Future<T>.completer （）** 方法已被删除。**future<T>** 直接扩展 **Handler<AsyncResult<T>**。您可以使用 **Future** 对象访问所有处理器方法。**Future** 对象也是处理程序。

4.3.3. 删除了 HTTP 客户端请求中的连接处理器方法

HttpClientRequest.connectionHandler （） 方法已被删除。使用 **HttpClient.connectionHandler （）** 方法调用应用中客户端请求的连接处理程序。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中使用 **HttpClientRequest.connectionHandler （）** 方法。

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

以下示例演示了如何在 Eclipse Vert.x 4 中使用新的 **HttpClient.connectionHandler （）** 方法。

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

4.4. VERTICLES 中的更改

本节介绍 verticles 中的更改。

4.4.1. create verticle 方法中的更新

在较早版本的 Eclipse Vert.x 中，**VerleFactory.createVerticle （）** 方法异步实例化verticle。从 Eclipse Vert.x 4 onward 中，该方法异步实例化verticle，并返回回调 **callable<Verticle** > 而不是单个验证实例。这种改进使应用调用此方法一次，并多次调用可返回调用，以创建多个实例。

以下代码演示了如何使用 Eclipse Vert.x 3.x 版本实例化。

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

以下代码演示了如何使用 Eclipse Vert.x 4 进行实例化。

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

4.4.2. factory 类和方法的更新

VerticleFactory 类已被简化。类不需要对标识符进行初始解析，因为工厂可以使用嵌套部署来部署 verticle。

如果您的现有应用程序使用工厂，在 Eclipse Vert.x 4 中，您可以更新代码，以便在承诺完成或失败时使用可调用的。可调用的可能是多次。

以下示例显示了 Eclipse Vert.x 3.x 应用程序中的现有因素。

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

以下示例演示了如何更新现有工厂以在 Eclipse Vert.x 4 中使用承诺。

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

如果您希望 factory 阻塞代码，请使用 **Vertx.executeBlocking （）** 方法。当工厂收到块代码时，它应该可以解决承诺并从承诺获取白色实例。

4.4.3. 删除了多线程 worker verticles

删除了多线程 worker verticle 部署选项。这个功能只能与 Eclipse Vert.x event-bus 一起使用。HTTP 等其他 Eclipse Vert.x 组件不支持该功能。

使用 unordered **Vertx.executeBlocking （）** 方法实现与多线程 worker 部署相同的功能。

4.5. 线程更改

本节介绍线程中的更改。

4.5.1. 非 Eclipse Vert.x 线程的上下文关联性

Vertx.getOrCreateContext （） 方法为每个非 Eclipse Vert.x 线程创建一个上下文。非 Eclipse Vert.x 线程在首次创建上下文时与上下文关联。在早期版本中，每次从非 Eclipse Vert.x 线程调用方法时，都会创建一个新上下文。

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

这个更改不会影响您的应用程序，除非应用程序隐式依赖于每个调用创建新上下文。

在以下示例中，因为每个阻塞代码在不同的上下文上调用了 n 块，所以 n 块会同时运行。

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

要在 Eclipse Vert.x 4 中获得相同的结果，您必须更新代码：

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

4.6. HTTP 的更改

本节介绍 HTTP 方法中的更改。

4.6.1. Eclipse Vert.x HTTP 方法中的通用更新

下面的部分论述了 Eclipse Vert.x HTTP 方法中的各种更新。

4.6.1.1. WebSocket的 HTTP 方法更新

WebSocket 中的更改有：

- 在方法名称中使用术语 **WebSocket** 不一致。方法名称具有不正确的大写，例如，Websocket 而非 **WebSocket**。**删除了在以下类中使用 WebSocket 的方法。使用正确大写的新方法。**

  - **HttpServerOptions 类中的以下方法已被删除。**

删除的方法	新方法
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()

  - **WebSocket 子协议的新方法使用 List<String > 数据类型而不是以逗号分隔的字符串**

来存储项目。

- HttpClientOptions 类中的以下方法已被删除。

删除的方法	替换方法
getTryUsePerMessageWebsocketCompression()	getTryUsePerMessageWebSocketCompression()
setTryUsePerMessageWebsocketCompression()	setTryUsePerMessageWebSocketCompression()
getTryWebsocketDeflateFrameCompression()	getTryWebSocketDeflateFrameCompression()
getWebsocketCompressionAllowClientNoContext()	getWebSocketCompressionAllowClientNoContext()
setWebsocketCompressionAllowClientNoContext()	setWebSocketCompressionAllowClientNoContext()
getWebsocketCompressionLevel()	getWebSocketCompressionLevel()
setWebsocketCompressionLevel()	setWebSocketCompressionLevel()
getWebsocketCompressionRequestServerNoContext()	getWebSocketCompressionRequestServerNoContext()
setWebsocketCompressionRequestServerNoContext()	setWebSocketCompressionRequestServerNoContext()

- HttpServer 类中的以下处理器方法已被删除。

弃用的方法	新方法
websocketHandler()	webSocketHandler()
websocketStream()	webSocketStream()

- WebsocketRejectedException 已被弃用。现在，方法会引发 UpgradeRejectedException。

- HttpClient webSocket （） 方法使用 Handler<AsyncResult<WebSocket>> 而不是

Handler &lt ;Throwable>。

- 使用 WebSocketConnectOptions 类中的方法也减少了将 HTTP 客户端连接到 WebSocket 的超载方法的数量。

- HttpServerRequest.upgrade （） 方法已被删除。此方法是同步的。

  使用新的方法 HttpServerRequest.toWebSocket （）。这个新方法是异步的。

  以下示例显示了在 Eclipse Vert.x 3.x 中使用同步方法。

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

  以下示例显示了在 Eclipse Vert.x 4 中使用异步方法。

```
// 4.x
server.requestHandler(req -> {
  Future<WebSocket> fut = req.toWebSocket();
  fut.onSuccess(ws -> {
  });
});
```

4.6.1.2. 设置 WebSocket 连接数量

在 Eclipse Vert.x 3.x 中，您可以使用 HTTP 客户端池大小在应用程序中定义最大 WebSocket 连接数。value accessor 方法 HttpClientOptions.maxPoolSize （） 用于获取和设置 WebSocket 连接。每个端点的默认连接数设置为 4。

以下示例演示了如何在 Eclipse Vert.x 3.x 中设置 WebSocket 连接。

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

但是，在 Eclipse Vert.x 4 中，没有池 WebSocket TCP 连接，因为连接在使用后关闭。应用为 HTTP 请求使用不同的池。使用值 accessor 方法 HttpClientOptions.maxWebSockets （） 获取和设

置 WebSocket 连接。每个端点的默认连接数设置为 50。

以下示例演示了如何在 Eclipse Vert.x 4 中设置 WebSocket 连接。

```
// 4.x
options.setMaxWebSockets(30); // Maximum connection is set to 30 for each endpoint
```

4.6.1.3. HttpMethod 作为一个接口可用

HttpMethod 可作为新接口使用。

在较早版本的 Eclipse Vert.x 中，Http Method 被声明为枚举的数据类型。作为一个枚举，它限制了 HTTP 的可扩展性。另外，它会阻止通过这个类型直接提供其他 HTTP 方法。您必须在服务器和客户端 HTTP 请求期间使用 HttpMethod.OTHER 值以及 rawMethod 属性。

如果您在交换机块中使用 HttpMethod enumerated 数据类型，您可以使用以下代码将应用程序迁移到 Eclipse Vert.x 4。

以下示例显示了 Eclipse Vert.x 3.x 版本的交换机块。

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

以下示例显示了 Eclipse Vert.x 4 中的交换机块。

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

您还可以在 Eclipse Vert.x 4 中使用以下代码。

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

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

如果您在应用程序中使用 HttpMethod.OTHER 值，请使用以下代码将应用程序迁移到 Eclipse Vert.x 4。

以下示例显示了 Eclipse Vert.x 3.x 版本中的代码。

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

以下示例显示了 Eclipse Vert.x 4 中的代码。

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

4.6.2. HTTP 客户端的更改

本节论述了 HTTP 客户端中的更改。

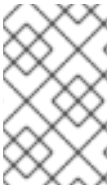
有以下类型的 Eclipse Vert.x 客户端可用：

Eclipse Vert.x web 客户端

当应用程序面向 web 时，使用 Eclipse Vert.x web 客户端。例如，REST、编码和解码 HTTP 有效负载，解释 HTTP 状态代码等。

Eclipse Vert.x HTTP 客户端

当应用程序用作 HTTP 代理时，请使用 Eclipse Vert.x HTTP 客户端。例如，作为 API 网关。HTTP 客户端已更新，并改进了 Eclipse Vert.x 4。



注意

Eclipse Vert.x web 客户端基于 Eclipse Vert.x HTTP 客户端。

4.6.2.1. 将应用程序迁移到 Eclipse Vert.x web 客户端

Web 客户端可从 Eclipse Vert.x 3.4.0 版本中找到。Eclipse Vert.x 4 中的 web 客户端没有更改。

客户端提供简化的 HTTP 交互和一些额外功能，如 HTTP 会话、JSON 编码和解码、响应 predicates，它们在 Eclipse Vert.x HTTP 客户端中不可用。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中使用 HTTP 客户端。

```
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();
```

以下示例演示了如何在 Eclipse Vert.x 3.x 和 Eclipse Vert.x 4 版本中将应用程序迁移到 web 客户端。

```
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. 将应用程序迁移到 Eclipse Vert.x HTTP 客户端

HTTP 客户端对 HTTP 交互具有精细的控制，并侧重于 HTTP 协议。

在 Eclipse Vert.x 4 中更新 HTTP 客户端并改进：

- 使用较少的交互简化 API
- 可靠的错误处理
- 支持 HTTP/1 的连接重置

HTTP 客户端 API 中的更新是：

- HttpClientRequest 中的方法，如 get （）、delete （） 和 put （） 已被移除。改为使用 HttpClientRequest> request (HttpMethod 方法 …)。
- 有可能在请求或响应时创建 HttpClientRequest 实例。例如，当客户端连接到服务器或连接时，会创建一个 HttpClientRequest 实例。

4.6.2.2.1. 发送简单请求

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中发送 GET 请求。

```
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();
```

以下示例演示了如何在 Eclipse Vert.x 4 中发送 GET 请求。

```
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
  }
});
```

您可以在新的 HTTP 客户端中看到错误处理效果更好。

以下示例演示了如何在 Eclipse Vert.x 4 中的 GET 操作中使用未来组成。

```
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
  }
});
```

未来组成提高了异常处理。示例检查状态代码是否为 200，否则返回了错误。

警告

当您将 HTTP 客户端与将来一起使用时，H ttpClientResponse （） 方法会在收到响应时立即发出缓冲区。为避免这种情况，请确保将来在事件循环上发生在事件循环上（如示例中所示），或者应该暂停和恢复响应。

4.6.2.2.2. 发送请求

在 Eclipse Vert.x 3.x 版本中，您可以使用 end （） 方法发送请求。

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

您还可以在请求中发送正文。

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

因为 HttpClientRequest 是一个 Writestream<Buffer >，所以您也可以使用管道来流传输请求。

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

在 Eclipse Vert.x 4 中，您可以使用 get （） 方法执行示例中显示的所有操作。您还可以使用新的 send （） 方法来执行这些操作。您可以将缓冲区、字符串或 ReadStream 作为输入传递给 send （） 方法。该方法返回 HttpClientResponse 实例。

```
// 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. 处理响应

HttpClientResponse 接口已经使用以下方法更新并改进：

body () 方法

body () 方法返回异步缓冲区。使用 body () 方法而不是 bodyHandler ()。

以下示例演示了如何使用 bodyHandler () 方法来获取请求正文。

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

以下示例演示了如何使用 body () 方法来获取请求正文。

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

end () 方法

当响应完全被完全接收或失败时，end () 方法会返回未来。该方法删除响应正文。使用此方法，而不使用 endHandler () 方法。

以下示例演示了如何使用 endHandler () 方法。

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

以下示例演示了如何使用 end （） 方法。

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

您还可以使用方法（如 onSucces （）、Compose （）、 bodyHandler （） 等）处理响应。以下示例演示了使用 onSuccess （） 方法处理响应。

以下示例演示了如何将 HTTP 客户端与 Eclipse Vert.x 3.x 版本的 result （） 方法搭配使用。

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

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

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

以下示例演示了如何将 HTTP 客户端与 Eclipse Vert.x 4 中的 result （） 方法搭配使用。

```
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 客户端的改进

本节论述了 HTTP 客户端的改进。

4.6.2.3.1. HTTP 客户端请求和响应方法将异步处理程序用作输入参数

HttpClient 和 HttpClientRequest 方法已更新，以使用异步处理程序。该方法使用 Handler<AsyncResult<HttpClientResponse> 作为输入，而不是 Handler<HttpClientResponse>。

在较早版本的 Eclipse Vert.x 中，HttpClient 方法 get Now （）、选项 Now （）和 headNow （）用于返回 HttpClient Request，您必须进一步发送请求。getNow （）、 optionsNow （） 和 headNow （） 方法已被删除。在 Eclipse Vert.x 4 中，您可以使用 Handler<AsyncResult<HttpClientResponse& gt; 直接发送包含所需信息的请求。

以下示例演示了如何在 Eclipse Vert.x 3.x 中发送请求。

- 执行 GET 操作：

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

- 使用缓冲正文进行 POST：

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

- 使用流正文 POST:

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

在 Eclipse Vert.x 4 中，您可以使用 请求 方法创建 HttpClientRequest 实例。这些方法可用于基本交互，例如：

- 发送请求标头
- HTTP/2 特定的操作，如设置 push 处理程序、设置流优先级、 pings 等。
- 创建 NetSocket 隧道

- 提供精细的写入控制
- 重置流
- 手动处理 100 个继续标头

以下示例演示了如何在 Eclipse Vert.x 4 中创建 HTTPClientRequest。

```
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. 从 HTTP 客户端请求删除了连接处理器方法

HttpClientRequest.connectionHandler （） 方法已被删除。使用 HttpClient.connectionHandler （） 方法调用应用中客户端请求的连接处理程序。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中使用 HttpClientRequest.connectionHandler （） 方法。

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

以下示例演示了如何使用新的 HttpClient.connectionHandler （） 方法。

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

4.6.2.3.3. 使用 net socket 方法进行 HTTP 客户端隧道

可以使用 HttpClientResponse.netSocket （） 方法创建 HTTP 隧道。在 Eclipse Vert.x 4 中，此方法已更新。

要获得用于请求连接的 net socket，请在请求中发送套接字处理程序。当收到 HTTP 响应标头时，调用该处理程序。套接字可以进行隧道连接，可以发送和接收缓冲区。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中获取连接的 net socket。

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

以下示例演示了如何在 Eclipse Vert.x 4 中获取连接的 net socket。

```
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 类中的新 send （） 方法

HttpClient 类中提供了一个新的 send （） 方法。

以下代码演示了如何在 Eclipse Vert.x 4 中发送请求。

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

4.6.2.3.5. httpHeaders 是一个接口，包含 MultiMap 方法

在 Eclipse Vert.x 4 中，HttpHeaders 是一个接口。在较早版本的 Eclipse Vert.x 中，HttpHeaders 是一个类。

在 HttpHeaders 界面中添加了以下新的 MultiMap 方法。使用这些方法创建 MultiMap 实例。

- MultiMap.headers()
- MultiMap.set (CharSequence name, CharSequence value)
- MultiMap.set （字符串名称、String 值）

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中创建 MultiMap 实例。

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

以下示例演示了如何在 Eclipse Vert.x 4 中创建 MultiMap 实例。

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

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

4.6.2.3.6. CaseIn sensitiveHeaders 类不再是公共的

CaseIn sensitiveHeaders 类不再是公共的。使用 MultiMap.caseInsensitiveMultiMap （） 方法创建一个具有不区分大小写的键的多映射实施。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中使用 CaseInsensitiveHeaders 方法。

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

以下示例演示了如何在 Eclipse Vert.x 4 中使用 MultiMap 方法。

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

或者

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

4.6.2.3.7. 检查服务器上运行的 HTTP 版本

在较早版本的 Eclipse Vert.x 中，只有在应用程序明确调用 HttpServerRequest.version （） 方法时，才会检查服务器上运行的 HTTP 版本。如果 HTTP 版本是 HTTP/1.x，该方法会返回 501 HTTP 状态，并关闭连接。

在向服务器发送请求前，从 Eclipse Vert.x 4 onward 中，通过调用 HttpServerRequest.version （） 方法自动检查服务器上的 HTTP 版本。当找到无效的 HTTP 版本时，方法会返回 HTTP 版本而不是抛出异常。

4.6.2.3.8. 请求选项中的新方法

在 Eclipse Vert.x 4 中， RequestOptions 类中提供了以下新方法：

- 标头
- FollowRedirects
- Timeout（超时）

以下示例演示了如何使用新方法。

```
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. 连接方法的更改

本节解释了连接方法中的更改。

4.7.1. 检查客户端是否需要身份验证

NetServerOptions.isClientAuthRequired （） 方法已被删除。使用 getClientAuth （）== ClientAuth.REQUIRED enumerated 类型来检查是否需要客户端身份验证。

以下示例演示了如何使用 switch 语句来检查是否需要验证客户端。

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

以下示例演示了如何使用检查 Eclipse Vert.x 4 中的客户端验证。

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

4.7.2. 升级 SSL 方法使用异步处理程序

NetSocket.upgradeToSsl （） 方法已更新为使用 Handler<AsyncResult> 而不是 Handler。该处理程序用于检查频道是否已成功升级到 SSL 或 TLS。

4.8. 日志更改

本节解释了日志记录中的更改。

4.8.1. 弃用的日志类和方法

日志记录 类 logger 和 LoggerFactory 及其方法已被弃用。这些日志类和方法将在以后的版本中删除。

4.8.2. 删除了 Log4j 1 日志记录器

Log4j 1 日志记录器不再可用。但是，如果要使用 Log4j 1 日志记录器，则可通过 SLF4J 使用。

4.9. ECLIPSE VERT.X REACTIVE EXTENSIONS (RX)中的更改

这部分论述了 Eclipse Vert.x 中的 Reactive Extensions (Rx)中的更改。Eclipse Vert.x 使用 RxJava 库。

4.9.1. 支持 RxJava 3

在 Eclipse Vert.x 4.1.0 中，支持 RxJava 3。

- 一个新的 rxified API 位于 io.vertx.rxjava3 软件包中。
- 与 Eclipse Vert.x JUnit5 集成由 vertx-junit5-rx-java3 绑定提供。

要升级到 RxJava 3，您必须进行以下更改：

- 在 pom.xml 文件中，&lt ;dependency& gt; 下的 RxJava 1 和 2 绑定从 vertx-rx-java 或 vertx-rx-java2 更改为 vertx-rx-java3。
- 在应用程序中，将 io.vertx.reactivex.* 的导入更新为 io.vertx.rxjava3.*。
- 在应用程序中，同时更新 RxJava 3 类型的导入。如需更多信息，请参阅 *RxJava 3 文档中的新内容*。

4.9.2. 从写入流中删除 Complete callback

WriteStreamSubscriber.onComplete （） 回调已被删除。如果 WriteStream 有待处理的数据流需要写入，则调用此回调。

在 Eclipse Vert.x 4 中，使用 callbacks WriteStreamSubscriber.onWriteStreamEnd （） 和 WriteStreamSubscriber.onWriteStreamError （）。这些回调在 WriteStream.end （） 完成后调用。

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

以下示例演示了如何从 Eclipse Vert.x 3.x 版本的 WriteStream 中创建适配器。

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

以下示例演示了如何使用 Eclipse Vert.x 4 中的新回调方法从 WriteStream 创建适配器：

```
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 配置的更改

以下部分介绍了 Eclipse Vert.x 配置的更改。

4.10.1. 检索配置的新方法

已删除方法 ConfigRetriever.getConfigAsFuture （）。使用方法 retriever.getConfig （） 替代。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中检索配置。

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

以下示例演示了如何在 Eclipse Vert.x 4 中检索配置。

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

4.11. JSON 的更改

本节描述了 JSON 中的更改。

4.11.1. Jackson 的封装

JSON 类中的所有实现 Jackson 类型的方法均已被删除。使用以下方法之一：

删除的字段/Methods	新方法
JSON.mapper （） 字段	DatabindCodec.mapper()
Json.prettyMapper() field	DatabindCodec.prettyMapper()
Json.decodeValue(Buffer, TypeReference<T>)	JacksonCodec.decodeValue(Buffer, TypeReference)
Json.decodeValue(String, TypeReference<T>)	JacksonCodec.decodeValue(String, TypeReference)

例如，使用以下代码：

- 使用 Jackson TypeReference 时：
  - 在 Eclipse Vert.x 3.x 版本中：

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

  - 在 Eclipse Vert.x 4 版本中：

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

- 引用 ObjectMapper ：
  - 在 Eclipse Vert.x 3.x 版本中：

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

  - 在 Eclipse Vert.x 4 版本中：

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

- 设置 ObjectMapper ：
  - 在 Eclipse Vert.x 3.x 版本中：

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

  - 从 Eclipse Vert.x 4 开始，您无法编写映射程序实例。您应该使用自己的静态映射程序或配置 Databind.mapper （） 实例。

4.11.2. 对象映射

在早期版本中，运行时需要 Jackson core 和 Jackson databind 依赖项。

从 Eclipse Vert.x 4 onward 中，只需要 Jackson 核心依赖项。

只有在对象映射 JSON 时，才需要 Jackson databind 依赖项。在这种情况下，您必须在 com.fasterxml.jackson.core:jackson-databind jar 中明确添加项目描述符中的依赖关系。

上述类型支持以下方法。

- Methods
  - JsonObject.mapFrom(Object)
  - JsonObject.mapTo (Class)
  - Json.decodeValue(Buffer, Class)
  - Json.decodeValue(String, Class)

- Json.encode(Object)
- Json.encodePrettily(Object)
- Json.encodeToBuffer(Object)

- 类型
  - JsonObject 和 JsonArray
  - 映射 和 列表
  - Number
  - 布尔值
  - Enum
  - byte[] 和 缓冲
  - 即时

只支持使用 Jackson bind 的方法：

- JsonObject.mapTo(Object)
- JsonObject.mapFrom(Object)

4.11.3. Base64 编码器更新为 JSON 对象和数组的 Base64URL

Eclipse Vert.x JSON 类型实现了 RFC-7493。在较早版本的 Eclipse Vert.x 中，实施会错误地使用 Base64 编码程序而不是 Base64URL。这个问题已在 Eclipse Vert.x 4 中修复，在 JSON 类型中使用 Base64URL encoder。

如果您要继续使用 Eclipse Vert.x 4 中的 Base64 编码器，您可以使用 旧的 配置标志。以下示例演示了如何在 Eclipse Vert.x 4 中设置配置标志。

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

如果您部分迁移了从 Eclipse Vert.x 3.x 到 Eclipse Vert.x 4 的迁移，则应用程序将在版本 3 和 4 上进行。在这种情况下，您可以使用以下实用程序将 Base64 字符串转换为 Base64URL。

```
public String toBase64(String base64Url) {
  return base64Url
    .replace('+', '-')
    .replace('/', '_');
}

public String toBase64Url(String base64) {
  return base64
    .replace('-', '+')
    .replace('_', '/');
}
```

在以下情况下必须使用工具方法：

- 在从 Eclipse Vert.x 3.x 版本迁移到 Eclipse Vert.x 4 时处理集成。
- 处理与使用 Base64 字符串的其他系统互操作性。

使用以下示例代码将 Base64URL 转换为 Base64 编码程序。

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

Base64 和 toBase64 Url 的帮助程序功能仅启用 JSON 迁移。如果您使用对象映射将 JSON 对象自动映射到应用程序中的 Java POJO，则必须创建一个自定义对象映射程序，将 Base64 字符串转换为 Base64URL。

以下示例演示了如何使用自定义 Base64 解码器创建对象映射程序。

```
// simple deserializer from Base64 to byte[]
class ByteArrayDeserializer extends JsonDeserializer<byte[]> {
  ByteArrayDeserializer() {
  }

  public byte[] deserialize(JsonParser p, DeserializationContext ctxt) {
    String text = p.getText();
    return Base64.getDecoder()
      .decode(text);
  }
}

// ...

ObjectMapper mapper = new ObjectMapper();

// create a custom module to address the Base64 decoding
SimpleModule module = new SimpleModule();
module.addDeserializer(byte[].class, new ByteArrayDeserializer());
mapper.registerModule(module);

// JSON to POJO with custom deserializer
mapper.readValue(json, MyClass.class);
```

4.11.4. 从信任选项中删除 JSON 转换器方法

TrustOptions.toJSON 方法已被删除。

4.12. ECLIPSE VERT.X WEB 的更改

以下部分介绍了 Eclipse Vert.x web 中的更改。

4.12.1. 将用户会话处理程序的功能组合到会话处理程序中

在较早版本的 Eclipse Vert.x 中，您必须在会话中指定 User SessionHandler 和 SessionHandler 处理程序处理程序。

为了简化这个过程，在 Eclipse Vert.x 4 中，UserSessionHandler 类已被删除，并在 SessionHandler 类中添加了其功能。在 Eclipse Vert.x 4 中，若要用于会话，您必须只指定一个处理程序。

4.12.2. 删除了 Cookie 接口

删除了以下 Cookie 接口：

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

使用 io.vertx.core.http.Cookie 接口。

4.12.3. Favicon 和 error 处理程序使用 Vertx 文件系统

FaviconHandler 和 ErrorHandler 中的 create 方法已更新。您必须在创建方法中传递 Vertx 实例对象。这些方法访问文件系统。传递 Vertx 对象可确保使用"Vertx"文件系统对文件进行一致的访问。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中使用创建方法。

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

以下示例演示了如何在 Eclipse Vert.x 4 中使用创建方法。

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

4.12.4. 访问模板引擎

使用 TemplateEngine.unwrap （） 方法访问模板引擎。然后，您可以将自定义和配置应用到模板。

用于获取和设置引擎配置的方法已弃用。改为使用 TemplateEngine.unwrap （） 方法。

- HandlebarsTemplateEngine.getHandlebars()

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

4.12.5. 删除了区域设置接口

io.vertx.ext.web.Locale 接口已被删除。使用 io.vertx.ext.web.LanguageHeader 接口。

4.12.6. 删除了可接受的区域设置方法

已删除 RoutingContext.acceptableLocales （） 方法。使用 RoutingContext.acceptableLanguages （） 方法。

4.12.7. 更新了挂载子路由器的方法

在较早版本的 Eclipse Vert.x 中，Router.mountSubRouter （） 方法错误地返回 路由器。这个问题已被解决，方法现在会返回一个 Route。

4.12.8. 删除了带有排除 JWT 身份验证处理的字符串的 create 方法

JWTAuthHandler.create (JWTAuth authProvider, String skip) 方法已被删除。使用 JWTAuthHandler.create (JWTAuth authProvider) 方法。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中创建 JWT 身份验证处理程序。

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

以下示例演示了如何在 Eclipse Vert.x 4 中创建 JWT 身份验证处理程序。

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

4.12.9. 删除了在 OSGi 环境中使用的创建处理器方法

在 Eclipse Vert.x 4 中，GGi 环境不再被支持。StaticHandler.create (String, ClassLoader) 方法已被删除，因为此方法已在 OSGi 环境中使用。

如果您在应用程序中使用了此方法，那么在 Eclipse Vert.x 4 中，您可以将资源添加到应用程序类路径或提供文件系统中的资源。

4.12.10. 删除网桥选项类

sockjs.BridgeOptions 类已被删除。改为使用新的 sockjs.SockJSBridgeOptions 类。sockjs.SockJSBridgeOptions 类包含配置事件总线网桥所需的所有选项。

新类的行为没有改变，但数据对象类的名称已改变。

在以前的版本中，当您使用 sockjs.BridgeOptions 类来添加新网桥时，会出现许多重复的配置。新类包含所有可能的通用配置，并删除重复的配置。

4.12.11. SockJS 套接字事件总线默认不注册集群事件

默认情况下，Sock JSSocket 不再注册集群事件总线使用者。如果要使用事件总线写入套接字，您必须在 SockJSHandlerOptions 中启用 writeHandler。当您启用 writeHandler 时，事件总线使用者默认设为 local。

```
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();
}));
```

您可以将事件总线消费者配置为集群。

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

4.12.12. 添加身份验证提供程序的新方法

SessionHandler.setAuthProvider (AuthProvider) 方法已弃用。改为使用 SessionHandler.addAuthProvider （） 方法。新的方法允许应用使用多个身份验证提供程序，并将会话对象链接到这些身份验证提供程序。

4.12.13. OAuth2 验证供应商创建方法需要 vertx 作为 constructor 参数

从 Eclipse Vert.x 4，OAuth2Auth.create (Vertx vertx) 方法需要 vertx 作为 constructor 参数。vertx 参数使用安全非阻塞随机数生成器来生成非保障，以确保应用程序的安全性。

4.13. ECLIPSE VERT.X WEB GRAPHQL 的更改

下面的部分论述了 Eclipse Vert.x Web GraphQL 中的更改。

重要

Eclipse Vert.x Web GraphQL 只作为技术预览提供。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关技术预览功能支持范围 的信息，请参阅红帽客户门户网站中的技术预览功能支持范围。

4.13.1. 更新了在多种语言（多语言）环境中支持的方法

现在，在 polyglot 环境中支持以下方法：* UploadScalar 现在是一个 factory，使用 UploadScalar.create （） 的方法替代。

- VertxBatchLoader 现在是一个工厂，使用方法 io.vertx.ext.web.handler.graphql.dataloader.VertxBatchLoader.create （）。

- VertxDataFetcher 现在是一个工厂，使用方法 io.vertx.ext.web.handler.graphql.schema.VertxDataFetcher.create （）。

- VertxPropertyDataFetcher 现在是一个工厂，使用方法 io.vertx.ext.web.handler.graphql.schema.VertxPropertyDataFetcher.create （）。

4.13.2. 在 Eclipse Vert.x Web GraphQL 中处理 POST 请求

在以前的版本中，Eclipse Vert.x Web GraphQL 处理程序可以处理自己的 POST 请求。它不需要 Eclipse Vert.x Web BodyHandler 处理请求。但是，这种实施容易受到 DDoS 攻击的影响。

从 Eclipse Vert.x 4 onward 中，需要处理 POST 请求 BodyHandler。在安装 Eclipse Vert.x Web GraphQL 处理程序前，您必须安装 BodyHandler。

4.14. MICROMETER 指标的更改

以下部分论述了 Micrometer 指标中的更改。

4.14.1. TCP 发送和接收字节以计数器的形式记录，其具有等效的 HTTP 请求和响应摘要

在以前的版本中，以下指标被记录为套接字的发行摘要。从 Eclipse Vert.x 4 onward 中，这些指标以计数器的形式记录，后者报告交换的数据量。

- net 客户端
  - vertx_net_client_bytes_read
  - vertx_net_client_bytes_written
- 净服务器

- vertx_net_server_bytes_read
- vertx_net_server_bytes_written

对于这些计数器，在 HTTP 中引入了等效的分布摘要。这些摘要用于收集有关请求和响应大小的信息。

- HTTP 客户端
  - vertx_http_client_request_bytes
  - vertx_http_client_response_bytes
- HTTP 服务器
  - vertx_http_server_request_bytes
  - vertx_http_server_response_bytes

4.14.2. 重命名指标

以下指标已被重命名。

旧指标名称	新指标名称	在组件中更新
*_connections	*_active_connections	网络客户端和服务器 HTTP 客户端和服务器
*_bytesReceived	*_bytes_read	Datagram 网络客户端和服务器 HTTP 客户端和服务器

旧指标名称	新指标名称	在组件中更新
***_bytesSent**	***_bytes_written**	Datagram 网络客户端和服务器 HTTP 客户端和服务器
***_requests**	***_active_requests**	HTTP 客户端 HTTP 服务器
***_requestCount_total**	***_requests_total**	HTTP 客户端 HTTP 服务器
***_responseTime_seconds**	***_response_time_seconds**	HTTP 客户端 HTTP 服务器
***_responseCount_total**	***_responses_total**	HTTP 客户端 HTTP 服务器
***_wsConnections**	***_active_ws_connections**	HTTP 客户端 HTTP 服务器
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**	
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**	

旧指标名称	新指标名称	在组件中更新

4.15. ECLIPSE VERT.X OPENAPI 的更改

在 Eclipse Vert.x 4 中，有新的模块 vertx-web-openapi 可用。仅用 vertx-web 使用此模块来开发合同驱动的应用程序。

新模块非常适合 Eclipse Vert.x Web Router。新模块需要以下 Eclipse Vert.x 依赖项：

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

新模块位于软件包 io.vertx.ext.web.openapi 中。

在 Eclipse Vert.x 4 中，支持旧的 OpenAPI 模块 vertx-web-api-contract，以促进迁移到新模块。建议您移至新模块 vertx-web-openapi 来使用新功能。

4.15.1. 新模块使用路由器构建程序

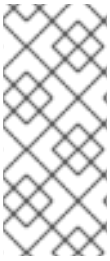
vertx-web-openapi 模块使用 RouterBuilder 来构建 Eclipse Vert.x Web 路由器。此路由器构建器与 vertx-web-api-contract 模块中的路由器 builer OpenAPI3RouterFactory 类似。

要开始使用 vertx-web-openapi 模块，可实例化 RouterBuilder。

```
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();
  }
});
```

您还可以使用未来实例化 RouterBuilder。

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



注意

vertx-web-openapi 模块使用 Eclipse Vert.x 文件系统 API 来加载文件。因此，您不必为 classpath 资源指定 /。例如，您可以在应用程序中指定 petstore.yaml。RouterBuilder 可识别您的类路径资源中的合同。

4.15.2. 新的路由器构建器方法

在大多数情况下，您可以使用新的 RouterBuilder 方法搜索和替换旧的 OpenAPI3RouterFactory 方法的使用。下表列出了一些旧方法和新方法示例。

旧的 OpenAPI3RouterFactory 方法	新的 RouterBuilder 方法
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()

使用以下语法访问解析的请求参数：

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

4.15.3. 处理安全性

在 Eclipse Vert.x 4 中，方法 RouterFactory.addSecurityHandler （） 和 OpenAPI3RouterFactory.addSecuritySchemaScopeValidator （） 不再可用。

改为使用 RouterBuilder.securityHandler （） 方法。此方法接受 io.vertx.ext.web.handler.AuthenticationHandler 作为处理程序。该方法自动识别 OAuth2Handler 并设置安全架构。

新的安全处理程序也实现了 OpenAPI 规格 中定义的操作。

4.15.4. 处理常见故障

在 vertx-web-openapi 模块中，以下失败处理程序不可用：您必须使用 Router.errorHandler (int, Handler) 方法设置失败处理程序。

'vertx-web-api-contract' 模块中的旧方法	vertx-web-openapi 模块中的新方法
routerFactory.setValidationFailureHandler(handler)	router.errorHandler (400、处理程序)
routerBuilder.setNotImplementedFailureHandler(handler)	router.errorHandler (501、处理程序)

4.15.5. 访问 OpenAPI 合同模型

在 Eclipse Vert.x 4 中，OpenAPI 合同没有映射到普通的旧的 Java 对象(POJO)。因此，不再需要额外的 swagger-parser 依赖项。您可以使用 getters 和解析器来检索合同的特定组件。

以下示例演示了如何使用单一操作来检索特定的组件。

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

以下示例演示了如何检索完整的合同。

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

以下示例演示了如何解决合同部分。

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

4.15.6. 在没有 OpenAPI 的情况下验证 Web 请求

在 vertx-web-api-contract 模块中，您可以使用 HTTPRequestValidationHandler 验证 HTTP 请求。您不必使用 OpenAPI 进行验证。

在 Eclipse Vert.x 4 中，验证 HTTP 请求使用 vertx-web-validation 模块。您可以导入这个模块，并在不使用 OpenAPI 的情况下验证请求。使用 ValidationHandler 验证请求。

4.15.7. Eclipse Vert.x web API 服务中的更新

vertx-web-api-service 模块已更新，并可与 vertx-web-validation 模块一起使用。如果您使用 vertx-web-openapi 模块，则不会更改 web 服务功能。

但是，如果您不使用 OpenAPI，则使用带有 vertx-web-validation 模块的 web 服务模块，则必须使用 RouteToEBServiceHandler 类。

```
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 模块不支持 vertx-web-api-contract。因此，当您升级到 Eclipse Vert.x 4 时，您必须将 Eclipse Vert.x OpenAPI 应用程序迁移到 vertx-web-openapi 模块。

第 5 章 微服务模式的更改

本节介绍微服务模式中的更改。

5.1. ECLIPSE VERT.X 断路器的更改

以下部分描述了 Eclipse Vert.x 断路器中的更改。

5.1.1. 删除了断路器中的执行命令方法

以下方法已从 CircuitBreaker 类中删除，因为它们无法用于将来。

删除的方法	替换方法
CircuitBreaker.executeCommand()	CircuitBreaker.execute()
CircuitBreaker.executeCommandWithFallback()	CircuitBreaker.executeWithFallback()

5.2. ECLIPSE VERT.X 服务发现中的更改

下面的部分论述了 Eclipse Vert.x 服务发现中的更改。

5.2.1. 从包含 ServiceDiscovery 参数的服务发现中删除方法

以下在服务发现中创建了方法，它使用 Handler<AmqpMessage > 作为参数被删除。这些方法无法用于未来的。

删除的方法	替换方法
ServiceDiscovery.create (…, Handler<ServiceDiscovery> completionHandler)	ServiceDiscovery.create(Vertx)
ServiceDiscovery.create (…, Handler<ServiceDiscovery> completionHandler)	ServiceDiscovery.create (Vertx, ServiceDiscoveryOptions)

5.2.2. 服务导入程序和导出器方法不再正常

ServiceDiscovery.registerServiceImporter （） 和 ServiceDiscovery.registerServiceExporter （） 方法不再是 fluent。该方法返回 Future<Void>。

5.2.3. Kubernetes 服务导入程序不再自动注册

vertx-service-discovery-bridge-kubernetes 添加 KubernetesServiceImporter 发现网桥。网桥从 Kubernetes 或 Openshift 将服务导入到 Eclipse Vert.x 服务发现。

从 Eclipse Vert.x 4，这个网桥不再自动注册。即使您已在 Maven 项目的类路径中添加了网桥，它也不会自动注册。

在创建 ServiceDiscovery 实例后，您必须手动注册该网桥。

以下示例演示了如何手动注册网桥。

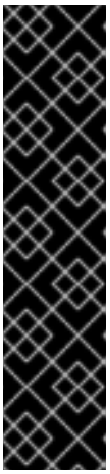
```
JsonObject defaultConf = new JsonObject();
serviceDiscovery.registerServiceImporter(new KubernetesServiceImporter(), defaultConf);
```

第 6 章 ECLIPSE VERT.X 身份验证和授权的更改

以下小节描述了 Eclipse Vert.x 身份验证和授权的更改。

Eclipse Vert.x 验证模块在 Eclipse Vert.x 4 中有重大更新。io.vertx.ext.auth.AuthProvider 接口被分成两个新接口：

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



重要

身份验证功能仅作为技术预览提供。技术预览功能不包括在红帽生产服务级别协议（SLA）中，且其功能可能并不完善。因此，红帽不建议在生产环境中使用它们。这些技术预览功能可以使用户提早试用新的功能，并有机会在开发阶段提供反馈意见。

有关技术预览功能支持范围 的信息，请参阅红帽客户门户网站中的技术预览功能支持范围。

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

在这个版本中，任何供应商都可以独立执行身份验证和授权。

6.1. 迁移身份验证应用程序

在结果级别上更改了验证机制。在以前的版本中，结果是 User 对象，它是特定于供应商的。在 Eclipse Vert.x 4 中，结果是对 io.vertx.ext.auth.User 的常见实现。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中验证用户。

```
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();
  }
});
```

以下示例演示了如何在 Eclipse Vert.x 4 中验证用户身份。

```
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. 迁移授权应用程序

授权是 Eclipse Vert.x 4 中的新功能。在之前的版本中，您只能检查用户是否有权在 User 对象上执行任务。这意味着该提供程序负责用户的身份验证和授权。

在 Eclipse Vert.x 4 中，User 对象实例不与特定的身份验证供应商相关联。因此，您可以使用不同的供应商验证并授权用户。例如，您可以使用 OAuth2 验证用户，并对 MongoDB 或 SQL 数据库执行授权检查。

以下示例演示了如何应用程序检查用户是否可以在 Eclipse Vert.x 3.x 版本中使用打印机 #1234。

```
// 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");
    }
  }
});
```

此授权适用于 JDBC 和 MongoDB。但是，它不适用于 OAuth2 等提供程序，因为该提供程序没有执行授权检查。从 Eclipse Vert.x 4 中，可以使用不同的提供程序执行此类授权检查。

```
// 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");
    }
  }
});
```

您可以检查对添加的角色、权限、逻辑操作、通配符和其他任何实施的授权。

6.3. 密钥管理的变化

在 Eclipse Vert.x 4 中，处理密钥有重大更新。最重要的更改是，当密钥加载时，公共缓冲区和私有缓冲之间没有区别。

以下的类已更新：

- io.vertx.ext.auth.KeyStoreOptions，用于 jce 密钥存储
- io.vertx.ext.auth.SecretOptions，用于处理对称 secret
- io.vertx.ext.auth.PubSecKeyOptions，用于处理公共 secret 密钥

下一节描述了关键管理中的更改。

6.3.1. Secret 选项类不再可用

SecretOptions 类不再可用。使用新的 PubSecKeyOptions 类来用于加密密钥。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中使用 SecretOptions 类方法。

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

以下示例演示了如何在 Eclipse Vert.x 4 中使用 PubSecKeyOptions 类的方法。

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

6.3.2. 公钥管理中的更新

在 Eclipse Vert.x 3.x 中，公钥密钥管理中的配置对象假定为：

- 密钥配置为密钥对。
- 密钥数据是 PKCS8 编码的字符串，没有标准分隔符。

以下示例演示了如何在 Eclipse Vert.x 3.x 中配置密钥对。

```
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----"));
```

在 Eclipse Vert.x 4 中，您必须指定公钥和私钥。

以下示例演示了如何在 Eclipse Vert.x 4 中配置密钥对。

```
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);
```

现在，您可以使用 PubSecKeyOptions 处理 X509 证书。

```
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. 密钥存储管理的变化

在 Eclipse Vert.x 3.x 中，KeyStoreOptions 假定密钥存储格式为 jceks，而存储的密码与密钥的密码相同。由于 jceks 是专有格式，建议使用标准的格式，如 JDK。

当在 Eclipse Vert.x 4 中使用 KeyStoreOptions 时，可以指定存储类型。例如，可以设置 PKCS11、P PKCS12 等存储类型。默认存储类型是 jceks。

在 Eclipse Vert.x 3.x 中，所有密钥存储条目都共享相同的密码，即密钥存储密码。在 Eclipse Vert.x 4 中，每个密钥存储条目都可以有专用密码。如果您不想为每个密钥存储条目设置密码，您可以将密钥存储密码配置为所有条目的默认密码。

以下示例演示了如何在 Eclipse Vert.x 3.x 中加载 jceks 密钥存储。

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

在 Eclipse Vert.x 4 中，假定默认格式是 JDK 配置的默认格式。格式是 Java 9 及更高版本中的 PKCS12。

以下示例演示了如何在 Eclipse Vert.x 4 中载入 jceks 密钥存储。

```
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. 弃用和删除的身份验证和授权方法

以下小节列出了为身份验证和授权已弃用和删除的方法。

6.4.1. 删除的身份验证和授权方法列表

删除了以下方法：

删除的方法	替换方法
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()	没有替换方法
AccessToken.refreshToken()	AccessToken.opaqueRefreshToken()
io.vertx.ext.auth.jwt.JWTOptions data object	io.vertx.ext.jwt.JWTOptions data object
Oauth2ClientOptions.isUseAuthorizationHeader()	没有替换方法
Oauth2ClientOptions.scopeSeparator()	没有替换方法

6.4.2. 弃用的身份验证和授权方法列表

以下方法已弃用：

弃用的方法	替换方法
OAuth2Auth.decodeToken()	**AuthProvider.authenticate()**
OAuth2Auth.introspectToken()	**AuthProvider.authenticate()**
OAuth2Auth.getFlowType()	没有替换方法
OAuth2Auth.loadJWK()	**OAuth2Auth.jwkSet()**
Oauth2ClientOptions.isUseAuthorizationHeader()	没有替换方法

6.4.3. 弃用的身份验证和授权类列表

以下类已弃用：

弃用的类	替换类
AbstractUser	使用 ' User.create (JsonObject)' 方法创建用户对象。
AuthOptions	没有替换类
JDBCAuthOptions	用于身份验证的 **JDBCAuthenticationOptions**，以及用于授权的 **JDBCAuthorizationOptions**
JDBCHashStrategy	没有替换类
OAuth2RBAC	**AuthorizationProvider**
Oauth2Response	建议使用 **WebClient** 类
KeycloakHelper	没有替换类

第 7 章 协议的更改

本节解释了网络协议中的更改。

7.1. ECLIPSE VERT.X GRPC 的更改

下面的部分论述了 Eclipse Vert.x gRPC 中的更改。

7.1.1. 新的 gRPC 编译器插件

在 Eclipse Vert.x 4 中，模块 protoc-gen-grpc-java 不再可用。这个模块是官方 gRPC 编译器的一个分叉。在较早版本的 Eclipse Vert.x 中，您必须使用这个 fork。此分叉由 Eclipse 项目维护。使用 fork 很复杂。

在以前的版本中，为了使用 gRPC，在 pom.xml 文件中添加以下详情。

```
  <!-- 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>
```

在 Eclipse Vert.x 4 中，提供了一个新的 gRPC 编译器插件。此插件使用官方 gRPC 编译器而不是 fork。要使用新的 gRPC 插件，请在 pom.xml 文件中添加以下详情。

```
    <!-- Vert.x 4.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 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. 迁移生成的代码

在 Eclipse Vert.x 4 中，使用新的编译器。当使用新的 gRPC 插件时，生成的代码不会使用相同的源文件写入。这是因为编译器不允许在其基础类上生成自定义代码。插件必须生成一个具有不同名称的新类来保存代码。

在较早版本的 Eclipse Vert.x 中，旧的 gRPC 插件会在同一源文件中写入生成的代码。

例如，如果您有以下描述符：

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

在 Eclipse Vert.x 3.x 中，代码会在 GreeterGrpc 类中生成。

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

在 Eclipse Vert.x 4 中，代码在 VertxGreeterGrpc 类中生成。

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

7.1.3. gRPC API 支持将来的

在 Eclipse Vert.x 4 中，gRPC API 支持将来的版本。gRPC 插件会生成混杂的 API。这些 API 使用标准 Eclipse Vert.x 输入和输出参数，这有助于创建标准 Eclipse Vert.x 应用程序。

以下示例显示了在 Eclipse Vert.x 3.x 中使用 promise。

```
// 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());
      }
  }
```

以下示例显示了在 Eclipse Vert.x 4 中使用未来。

```
// 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 的更改

下面的部分论述了 Eclipse Vert.x MQTT 中的更改。

7.2.1. MQTT 客户端中的一些流畅方法返回未来的

MqttClient 类中的一些流畅方法返回 Future，而不是 fluent。例如，MqttClient. connect （）、MqttClient.disconnect （）、MqttClient.disconnect （）、MqttClient.publish （） 返回 Eclipse Vert.x 4 中的未来。

以下示例显示了在 Eclipse Vert.x 3.x 版本中使用 publish （） 方法。

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

以下示例显示了在 Eclipse Vert.x 4 版本中使用 publish （） 方法。

```
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 消息返回缓冲

Mqtt will data 对象将字符串消息包装为 Eclipse Vert.x 缓冲，而不是字节阵列。

7.2.3. 从 MQTT 中删除已弃用的 Mqtt will 和 authorization 方法

删除了以下 MQTT 方法：

删除的方法	替换方法
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 SERVICE PROXY 的更改

以下部分描述了服务代理中的更改。

7.3.1. 使用服务代理代码生成器

ServiceProxyProcessor 类已被删除。

要使用服务代理代码生成器，您必须将 vertx-codegen 和 classpath 中的 processor classifier 导入：

```
<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>
```

服务代理重用 io.vertx.codegen.CodeGenProcessor from vertx-codegen 中的 io.vertx-codeor 启动服务代理和处理程序的代码生成。

第 8 章 客户端组件的变化

本节介绍 Eclipse Vert.x 客户端中的更改。

8.1. ECLIPSE VERT.X KAFKA 客户端的更改

下面的部分论述了 Eclipse Vert.x Kafka 客户端中的更改。

8.1.1. AdminUtils 类不再可用

AdminUtils 类不再可用。使用新的 KafkaAdminClient 类，在 Kafka 集群上执行管理操作。

8.1.2. 冲刷方法使用异步处理程序

KafkaProducer 类中的 flush 方法使用 Handler<AsyncResult<Void>> 而不是 Handler<Void>。

8.2. ECLIPSE VERT.X JDBC 客户端的更改

从 Eclipse Vert.x 4 中，JDBC 客户端支持 SQL 客户端。SQL common 模块也已合并到 JDBC 客户端中，即 io.vertx:vertx-sql-common 合并于 io.vertx:vertx-client 模块。您必须删除 io.vertx:vertx-sql-common 依赖项文件，因为 io.vertx:vertx-jdbc-client 将包含它。通过合并 SQL 通用客户端，所有数据库 API 都已合并到 JDBC 客户端中。

在 Eclipse Vert.x 4 中，SQL 客户端已被更新，使其包含以下客户端：

- 重新主动 PostgreSQL 客户端。在较早版本的中，它包括了被动 PostgreSQL 客户端。
- 重新主动 MySQL 客户端
- 重新主动 DB2 客户端
- 继续包括被动 PostgreSQL 客户端。这个客户端也包括在 Eclipse Vert.x 3.x 版本中。

- 现有的 JDBC 客户端现在同时包含 JDBC 客户端 API 和 SQL 客户端 API

被动实施使用数据库网络协议。这使得资源效率更高。

对数据库的 JDBC 调用将阻止调用。JDBC 客户端使用 worker 线程来使这些调用非阻塞。

以下部分介绍了 Eclipse Vert.x JDBC 客户端中的更改。

8.2.1. 创建池

在 Eclipse Vert.x 4 中，您可以使用 JDBC 客户端 API 创建池。在早期版本中，您只能创建客户端。您不能创建池。

以下示例演示了如何在 Eclipse Vert.x 3.x 中创建客户端。

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

以下示例演示了如何在 Eclipse Vert.x 4 中创建池。

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

注意

虽然 Eclipse Vert.x 3.x API 在 Eclipse Vert.x 4 中被支持，但建议您使用应用程序中的新的 JDBC 客户端 API。

池允许您执行简单的查询。您不需要管理简单查询的连接。但是，对于复杂的查询或多个查询，您必须管理您的连接。

以下示例演示了如何管理 Eclipse Vert.x 3.x 中的查询的连接。

```
// 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
  }
});
```

以下示例演示了如何管理 Eclipse Vert.x 4 中的查询连接。

```
// 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. 支持 Typsesafe 配置

您可以使用 jsonConfig 进行配置。但是，使用 jsonConfig 有时可能会导致错误。为避免这些错误，JDBC 客户端引入了 Typesafe Config。

以下示例显示了 Typesafe Config 的基本结构。

```
// 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)
    );
```

注意

要使用 Typesafe Config，您必须在项目中包含 agroal 连接池。该池不公开许多配置选项，并使配置易于使用。

8.2.3. 运行 SQL 查询

本节演示了如何在 JDBC 客户端中运行查询。

8.2.3.1. 运行一个 shot 查询

以下示例演示了如何在 Eclipse Vert.x 3.x 中在不管理连接的情况下运行查询。

```
// 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
  }
});
```

以下示例演示了如何在 Eclipse Vert.x 4 中运行查询，而不管理连接。

```
// 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. 在受管连接上运行查询

以下示例演示了如何在 Eclipse Vert.x 4 中对受管连接运行查询。

```
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. 支持存储的步骤

存储的步骤在 JDBC 客户端中受到支持。

以下示例演示了如何在 Eclipse Vert.x 3.x 中传递 IN 参数。

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

以下示例演示了如何在 Eclipse Vert.x 4 中传递 IN 参数。

```
// 4.x
client
  .preparedQuery("{call new_customer(?, ?)}")
  .execute(Tuple.of("Paulo", "Lopes"))
  .onSuccess(rows -> {
    ...
  });
```

在 Eclipse Vert.x 3.x 中，由于可用类型，将 IN 和 OUT 参数结合使用的支持会非常有限。在 Eclipse Vert.x 4 中，池是安全的，可以处理 IN 和 OUT 参数的组合。您还可以在应用程序中使用 INOUT 参数。

以下示例显示了在 Eclipse Vert.x 3.x 中处理参数。

```
// 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!
    }
});
```

以下示例显示了在 Eclipse Vert.x 4 中处理参数。

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

在 JDBC 客户端中，数据类型已更新。

- 对于类型为 OUT 的参数，您可以指定其返回类型。在示例中，OUT 参数指定为 类型 VARCHAR，这是 JDBC 常量。

- 这类类型不受 JSON 限制的限制。现在，您可以在类型名称中使用数据库特定类型而不是文本常量。

8.3. ECLIPSE VERT.X 邮件客户端的更改

下面的部分论述了 Eclipse Vert.x 邮件客户端的更改。

8.3.1. MailAttachment 作为一个接口提供

从 Eclipse Vert.x 4 开始，MailAttachment 作为一个接口可用。它可让您使用流中的邮件附加功能。在早期的 Eclipse Vert.x 版本中，邮件 附加程序作为一个类和邮件附件表示，以数据对象表示。

8.3.2. 邮件配置接口扩展了网络客户端选项

MailConfig 接口扩展 NetClientOptions 接口。由于此扩展，邮件配置也支持 NetClient 的代理设置。

8.4. ECLIPSE VERT.X AMQP 客户端的更改

下面的部分论述了 Eclipse Vert.x AMQP 客户端中的更改。

8.4.1. 删除了包含 AmqpMessage 参数的 AMQP 客户端中的方法

将 Handler<AmqpMessage> 用作参数的 AMQP 客户端方法已被删除。在较早版本的中，您可以在 ReadStream<AmqpMessage > 上设置此处理程序。但是，如果您将应用程序迁移到将来使用，则无法使用此类方法。

删除的方法	替换方法
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 客户端的更改

下面的部分论述了 Eclipse Vert.x MongoDB 客户端中的更改。

8.5.1. 从 MongoDB 客户端中删除的方法

的方法已从 MongoClient 类中删除。

删除的方法	替换方法
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()
MongoClient.removeOneWithOptions()	MongoClient.removeDocumentsWithOptions()

8.6. EVENTBUS JAVASCRIPT 客户端的变化

在 Eclipse Vert.x 4 中，EventBus JavaScript 客户端模块位于一个新的位置。您必须更新构建系统，以从新位置使用该模块。

在 Eclipse Vert.x 3.x 中，事件总线 JavaScript 客户端位于不同的位置，例如：

- Maven Central
- NPM
- Bower.io
- CDNJS

- webjars

在 Eclipse Vert.x 4 中，JavaScript 客户端仅在 npm 中提供。EventBus JavaScript 客户端模块可以从以下位置访问：

- CDN
- Npm 软件包

在构建脚本中使用以下代码来访问该模块。

- JSON 脚本

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

- XML 脚本

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

8.6.1. JavaScript 客户端的版本

在 Eclipse Vert.x 4 之前，每个 Eclipse Vert.x 发行版本都包含 JavaScript 客户端的新版本。

但是，从 Eclipse Vert.x 4 onward，只有在客户端有变化时才会在 npm 中提供一个新的 JavaScript 客户端版本。您不需要为每个 Eclipse Vert.x 版本更新客户端应用程序，除非有版本更改。

8.7. ECLIPSE VERT.X REDIS 客户端的更改

在 Eclipse Vert.x 4 中，使用 Redis 类用于 Redis 客户端。class RedisClient 不再可用。

注意

为了帮助您将应用程序从 RedisClient 迁移到 Redis 类，可以使用一个帮助程序类 RedisAPI。RedisAPI 可让您复制与 RedisClient 类类似的功能。

新类包含协议和 Redis 服务器功能中的所有增强功能。使用新类来：

- 使用所有 Redis 命令
- 连接到单一服务器
- 连接到启用了 Redis Sentinel 的高可用性服务器
- 连接到 Redis 的集群配置
- 在 Redis 扩展中执行请求
- 与 RESP2 和 RESP3 服务器协议服务器通信

8.7.1. 将现有 Redis 客户端应用程序迁移到新客户端

您可以将现有应用程序直接迁移到新的 Redis 客户端，或者在两个步骤中使用帮助程序类 RedisAPI 来迁移应用程序。

在迁移应用程序前，您必须创建客户端。

8.7.1.1. 创建客户端

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中创建 Redis 客户端。

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

以下示例演示了如何在 Eclipse Vert.x 4 中创建 Redis 客户端。

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

在 Eclipse Vert.x 4 中，客户端使用以下标准连接字符串语法：

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

8.7.1.2. 将应用程序迁移到 RedisAPI

使用"RedisAPI"，您现在可以决定如何管理连接：

- 您可以让客户端管理使用池的连接。

或者

- 您可以通过请求新连接来控制连接。完成后，您必须确保关闭或返回连接。

您必须创建客户端，然后更新应用程序以处理请求。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中创建客户端后处理请求。

```
// 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());
```

```
      }
    });
  }
});
```

以下示例演示了如何在 Eclipse Vert.x 4 中创建客户端后处理请求。这个示例使用列表来设置键值对，而不是硬编码选项。有关该命令可用的参数的更多信息，请参阅 Redis SET 命令。

```
// 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. 将应用程序直接迁移到 Redis 客户端

当您直接迁移到新的 Redis 客户端时：

- 您可以使用所有新的 Redis 命令。
- 您可以使用扩展。
- 您可以将一些从帮助程序类转换到新的客户端，从而改进应用程序的性能。

您必须创建客户端，然后更新应用程序以处理请求。

以下示例演示了如何在 Eclipse Vert.x 3.x 版本中创建客户端后设置和获取请求。

```
// 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());
      }
    });
  }
});
```

以下示例演示了如何在 Eclipse Vert.x 4 中创建客户端后处理请求。

```
// 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());
        }
      });
    }
});
```

在 Eclipse Vert.x 4 中，所有交互都使用 send (Request) 方法。

8.7.1.4. 迁移响应

在 Eclipse Vert.x 3.x 中，用来硬编码所有已知的命令 till Redis 5 的客户端也根据 命令键入响应。

在新客户端中，命令不会硬编码。响应是 Response 类型。新的线路协议具有更多类型范围。

在较旧的客户端中，响应是以下类型：

- null

- Long
- 字符串
- JsonArray
- JsonObject （适用于 INFO 和 HMGET 数组响应）

在新客户端中，响应是以下类型：

- null
- 响应

Response 对象具有类型转换器。例如，转换器，例如：

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

如果收到的数据不是请求的类型，则类型转换器将其转换为最接近的数据类型。当无法转换到特定类型时，会引发 UnsupportedOperationException。例如，无法从 String 转换到 List 或 Map。

您还可以处理集合，因为 Response 对象实现了可 Iterable 接口。

以下示例演示了如何执行 MGET 请求。

```
// 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 客户端中的更新

本节论述了 Redis 客户端中的更改。

8.7.2.1. 从 Redis 角色和节点选项中删除已弃用的术语 "slave"

在 Redis 角色和节点选项中，已弃用的术语"slave"已被"replica"替代。

角色

以下示例显示了在 Eclipse Vert.x 3.x 版本中使用 SLAVE 角色。

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

以下示例显示了在 Eclipse Vert.x 4 中使用 REPLICA 角色。

```
// After (4.x)
Redis.createClient(
      rule.vertx(),
      new RedisOptions()
        .setType(RedisClientType.SENTINEL)
        .addConnectionString("redis://localhost:5000")
        .setMasterName("sentinel7000")
        .setRole(RedisRole.REPLICA));
```

节点选项

以下示例显示了在 Eclipse Vert.x 3.x 版本中使用节点类型 RedisSlaves。

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

以下示例显示了在 Eclipse Vert.x 4 中使用节点类型 RedisReplicas。

```
// After (4.x)
options.setUseReplicas(RedisReplicas);
```

第 9 章 集群的更改

本节解释了集群中的更改。

9.1. 从选项类中删除集群的标志

在 Eclipse Vert.x 应用程序中指定、get 和 set 集群的方法和布尔值已从 VertxOptions 和 EventBusOptions 类中删除。

9.2. INFINISPAN 集群管理器的更改

以下部分介绍了 Infinispan 集群管理器的更改。

9.2.1. 自定义配置中的更新

Infinispan 集群管理器基于 Infinispan 12。

在 Eclipse Vert.x 4 中，集群 SPI 已被重新设计。订阅数据模型已改变。因此，Eclipse Vert.x 3.x 节点和 Eclipse Vert.x 4 节点无法在同一个 Infinispan 集群中添加。

Eclipse Vert.x 应用程序不会受到这个变化的影响，因为 EventBus 和 SharedData API 保持不变。

如果您在 Eclipse Vert.x 3.x 应用程序中有一个自定义 Infinispan 配置文件：

- 将 __vertx.subs 缓存类型更改为复制，而不是分布式。
- 添加复制的缓存 __vertx.nodeInfo。

```
<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>
```

如果您在 Openshift 上运行 Eclipse Vert.x 集群，则不再需要 infinispan-cloud JAR。JAR 已从构建文件的 dependencies 部分中移除。此 JAR 中提供的配置文件现在包含在 infinispan-core JAR 中。

9.3. 迁移集群

确定您的代码库的迁移策略非常重要。这是因为您无法在单个集群中将 Eclipse Vert.x 3.x 节点和 Eclipse Vert.x 4 节点添加到单个集群中，原因如下：

- 集群管理器升级 - 集群管理器中的 Major 版本升级会阻止向后兼容性。
- 订阅数据更改 - Eclipse Vert.x 更改了存储在集群管理器中的 EventBus 订阅数据的格式。
- 传输协议更改 - Eclipse Vert.x 已更改了集群中消息传输协议中的一些字段。

如果您的 Eclipse Vert.x 集群用于单个应用程序或某些密切相关的微服务，您可以一次将整个代码库迁移到新集群中。

但是，如果您无法一次性迁移代码库，请使用本节中的建议将 Eclipse Vert.x 3.x 代码库迁移到 Eclipse Vert.x 4。

9.3.1. 分割集群

如果您的集群已为其应用程序部署了不同的团队，您可以考虑将 Eclipse Vert.x 3.x 集群拆分为较小的集群。请注意，在分割集群后，独立的组件将无法使用集群功能进行通信。您可以使用以下组件分离集群：

- EventBus 请求和回复 - HTTP 或 RESTful Web 服务, gRPC
- EventBus 发送并发布 - Messaging 系统 Postgres LISTEN 和 NOTIFY、Red Hat Redis Pub 和 Sub
- 共享数据 - Redis、Infinispan

在分割集群后，每个团队在就绪或者需要时可以移至 Eclipse Vert.x 4。

9.3.2. Using Eclipse Vert.x EventBus Link

如果您无法分割集群，则使用 Vert.x EventBus Link 逐步迁移您的代码库。

Vert.x EventBus Link 是一个将 Eclipse Vert.x 3.x 集群事件Bus 连接到 Eclipse Vert.x 4 集群事件 Bus 的工具。

警告

不支持迁移共享数据 API，即、映射、计数器和锁定。

该工具创建一个实现 EventBusLink 接口的 EventBus Link 对象。在每个集群至少一个节点上创建了一个 EventBusLink 实例。该实例通过提供一组地址及其行为取决于消息模式：

- *触发并忘记 请求和回复* - 邮件发送到远程集群。
- *publish* - 消息发送到此集群和远程集群。

Eclipse Vert.x EventBus Link 创建一个 WebSocket 服务器来接收消息，并使用 WebSocket 客户端来发送它们。

详情请查看 部分 入门和使用。

第 10 章 ECLIPSE VERT.X 中的其它更改

下面的部分论述了 Eclipse Vert.x 4 中的各种更改。

10.1. 删除 STARTER 类

Starter 类已被删除。改为使用 Launcher 类来启动 Eclipse Vert.x 应用，而不带有 main （） 方法。

10.2. JAVA 8 的隔离部署

Eclipse Vert.x 4 支持 Java 11。这个 Java 版本不支持隔离类加载。在 Eclipse Vert.x 4 中，Java 8 支持隔离类加载。

10.3. 从 ECLIPSE VERT.X 上下文中删除了 HOOK 方法

方法 Context.addCloseHook （） 和 Context.removeCloseHook （） 方法已从 Context 类中删除。这些方法已移到内部接口 InternalContext 中。

10.4. 从选项中删除克隆方法

方法 KeyCertOptions.clone （）、TrustOptions.clone （） 和 SSLEngineOptions.clone （） 已被删除。改为使用 methods KeyCertOptions.copy （）、TrustOptions.copy （） 和 SSLEngineOptions.copy （）。

10.5. 从选项中删除等于和哈希码方法

VertxOptions.equals （） 和 VertxOptions.hashCode （） 方法已被删除。

10.6. 检查文件缓存的新方法

VertxOptions.fileResolverCachingEnabled （） 方法已被删除。使用 FileSystemOptions.isFileCachingEnabled （） 方法检查文件缓存是否已启用了解析类路径。

10.7. 服务提供商接口(SPI)指标

Metrics.isEnabled （） 方法已被删除。服务提供商接口(SPI)指标将返回一个 null 对象，以表示未启

用指标。

10.8. 删除池缓冲方法

pooled buffer 方法 TCPSSLOptions.isUsePooledBuffers （） 和 TCPSSLOptions.setUsePooledBuffers （） 已被删除。

10.9. 创建没有共享数据源的客户端的方法

使用以下新方法创建与其他客户端没有共享数据源的客户端。这些方法维护自己的数据源。

弃用的方法	新方法
MongoClient.createNonShared()	MongoClient.create()
JDBCClient.createNonShared()	wJDBCClient.create()
CassandraClient.createNonShared()	CassandraClient.create()
MailClient.createNonShared()	MailClient.create()

10.10. ECLIPSE VERT.X JUNIT5 的更改

下一节介绍 Eclipse Vert.x JUnit5 中的更改。

10.10.1. 支持 vertx-core 模块和扩展中的更新

vertx-core 模块已更新，以使用服务提供商接口进行参数注入。此更改导致 JUnit5 中的以下更新：

- 在需要创建它的任何参数前，您必须调用 Vertx 参数。例如，在注入 WebClient 时。
- vertx-junit5 模块仅支持 vertx-core 模块。
- reactiverse-junit5-extensions 模块主机包含额外参数类型的扩展，如 WebClient。
-

RxJava 1 和 2 绑定现在可作为 vertx-junit5-rx-java 和 vertx-junit5-rx-java2 模块包括在 vertx-junit5-extensions 存储库中。

从 Eclipse Vert.x 4.1.0，RxJava 3 绑定 vertx-junit5-rx-java3 可用。

10.10.2. 在 Eclipse Vert.x 文本上下文中弃用成功和失败的方法

VertxTestContext.succeeding （） 和 VertxTestContext.failing （） 方法已被弃用。使用 VertxTestContext.succeeding ThenComplete （） 和 VertxTestContext.failing ThenComplete （） 方法替代。