链滴

vert.x 4 使用插件完善你的项目

作者：lizhongyue248
原文链接：https://ld246.com/article/1580550244262
来源网站：链滴

vert.x 一直是一个维护非常积极的项目，从诞生至今一直如此。现在他的 GitHub 上面 open 的 issues 也只有一百多个，保持积极处理的状态，而不是堆积一堆。并在在提出相关 issue 以后，都会积极的应以及处理。

在 vert.x 中，它的一大特点就是 Verticle。每一个 Verticle 都有自己的功能，各司其职。并且，他们可以是事件驱动的。今天我们的主要任务就是改善我们在官网上面生成下来的 vert.x 项目，通过使用件来让我们的项目运行简单、部署简单。

我们主要用到如下两款插件：

1. vertx-plugin：这是官方提供的 vert.x 的 gradle 插件。**主要功能是提供完整的依赖管理，以及轻整合shadow ,application 等构建工具。**

2. vertx-boot：这虽然不是官方提供的，但是开发者依旧是核心的官方人员。**主要功能是通过配置文的方式来配置启动我们的Verticle，不需要在代码中写死或者命令行传参。**

目前（2020-2-1）这两款插件一直在积极的维护中。通过这两款插件，能够极大的让我们的 vert.x 用更加灵活，更容易部署。

当然，现在我开始向 Koltin 转型，所以以下的配置均是使用 Kotlin DSL 进行书写的 Gradle 文件。

所以主要分为三部分：

1. 修改项目初始化的文件，将 Groovy DSL 的 Gradle 文件 修改为 Kolitn DSL 的 Gradle 文件。
2. 整合 vertx-plugin
3. 整合 vertx-boot

2020.2.6 更新：vert.x 中移除了 deploy 的几个关于类加载的参数，参见 ISSUE。vertx-boot 还没对其更新后的参数进行完整性支持，所以如果使用 vert.x 4 版本，

- 对于 Kotlin 会造成 NoSuchMethodError 错误。

- 对于 Java 会提示方法过期。

在这之前

我们需要从 vert.x starter 上面初始化我们的项目。目前提供的有如下四个版本：

- 3.8.5
- 3.7.1
- 4.0.0-milestone4
- 4.0.0-SNAPSHOT

我并不是用在生产环境上的，所以大可以使用 4.0.0 版本。而我不选择使用里程碑版本，使用快照版即可。版本间的具体差别请自行谷歌。值得注意的是，3.x.x 版本到 4.x.x 版本是大版本更新，有了很的改变。

同时选择使用 Kotlin 、Gradle 来创建项目，组件选择使用 Web，其他随意，高级选项可以选择 Jdk 1。

使用 Kotlin DSL 改造项目

下载初始化后的项目以后，我们的 build.gradle 大概是这样子的：

```
plugins {
  id 'org.jetbrains.kotlin.jvm' version '1.3.20'
  id 'application'
  id 'com.github.johnrengelman.shadow' version '5.0.0'
}

group = 'cn.edu.gzmu.center'
version = '1.0.0-SNAPSHOT'

repositories {
  maven { url 'http://maven.aliyun.com/nexus/content/groups/public/' }
  maven {
    url 'https://oss.sonatype.org/content/repositories/snapshots'
    mavenContent {
      snapshotsOnly()
    }
  }
  mavenCentral()
  jcenter()
}

ext {
  kotlinVersion = '1.3.20'
  vertxVersion = '4.0.0-SNAPSHOT'
  junitJupiterEngineVersion = '5.4.0'
}

application {
  mainClassName = 'io.vertx.core.Launcher'
```

```
}


def mainVerticleName = 'cn.edu.gzmu.center.MainVerticle'
def watchForChange = 'src/**/*'
def doOnChange = './gradlew classes'

dependencies {
  implementation "io.vertx:vertx-web-client:$vertxVersion"
  implementation "io.vertx:vertx-auth-jwt:$vertxVersion"
  implementation "io.vertx:vertx-auth-oauth2:$vertxVersion"
  implementation "io.vertx:vertx-unit:$vertxVersion"
  implementation "io.vertx:vertx-web:$vertxVersion"
  implementation "io.vertx:vertx-config:$vertxVersion"
  implementation "io.vertx:vertx-config-yaml:$vertxVersion"
  implementation "io.vertx:vertx-pg-client:$vertxVersion"
  implementation "io.vertx:vertx-lang-kotlin-coroutines:$vertxVersion"
  implementation "io.vertx:vertx-consul-client:$vertxVersion"
  implementation "io.vertx:vertx-lang-kotlin:$vertxVersion"

  testImplementation "io.vertx:vertx-junit5:$vertxVersion"
  testRuntimeOnly "org.junit.jupiter:junit-jupiter-engine:$junitJupiterEngineVersion"
  testImplementation "org.junit.jupiter:junit-jupiter-api:$junitJupiterEngineVersion"
}

compileKotlin {
  kotlinOptions.jvmTarget = '1.8'
}

compileTestKotlin {
  kotlinOptions.jvmTarget = '1.8'
}


shadowJar {
  archiveClassifier.set('fat')
  manifest {
    attributes 'Main-Verticle': mainVerticleName
  }
  mergeServiceFiles {
    include 'META-INF/services/io.vertx.core.spi.VerticleFactory'
  }
}

test {
  useJUnitPlatform()
  testLogging {
    events 'PASSED', 'FAILED', 'SKIPPED'
  }
}

run {
  args = ['run', mainVerticleName, "--redeploy=$watchForChange", "--launcher-class=$mainC
assName", "--on-redeploy=$doOnChange"]
```

```
}
```

我们的第一步，将它改成我们 Kotlin DSL 写的 Gradle，我们一块一块的来。

在这之前要将文件build.gradle 重命名为 build.gradle.kts。

plugins

plugin 部分很简单，只是单双引号的函数调用的问题：

```
plugins {
  application
  kotlin("jvm") version "1.3.20"
  id("com.github.johnrengelman.shadow") version "5.0.0"
}
group = "cn.edu.gzmu.center"
version = "1.0.0-SNAPSHOT"
```

repository

repositories 也是如此，不过我们不用让他拉取快照了。

```
repositories {
  maven("http://maven.aliyun.com/nexus/content/groups/public/")
  maven("https://oss.sonatype.org/content/repositories/snapshots")
  mavenCentral()
  jcenter()
}
```

ext

对于 Gradle 的 extra 部分较为复杂，在 Koltin DSL 中，设置与获取不太优雅

```
ext {
    set("vertxVersion", "4.0.0-SNAPSHOT")
}
//......
// 使用
implementation "io.vertx:vertx-auth-oauth2:${extra["vertxVersion"]}"
```

另外一种方式是通过委托

```
val vertxVersion by extra { "4.0.0-SNAPSHOT" }

//......
// 使用
implementation "io.vertx:vertx-auth-oauth2:$vertxVersion"
```

我们姑且使用第二种方式。

```
// 变量直接把 def 改成 val，单引号改成双引号就可以了。
val mainVerticleName = "cn.edu.gzmu.center.MainVerticle"
```

```
val kotlinVersion by extra { "1.3.20" }
val vertxVersion by extra { "4.0.0-SNAPSHOT" }
// 加一个日志进去
val log4j2Version by extra { "2.13.0" }
val junitJupiterEngineVersion by extra { "5.4.0" }
```

application

这里就是单双引号的问题

```
application {
  mainClassName = "io.vertx.core.Launcher"
}
```

dependencies

这里就是单双引号的问题，同时要加上括号

```
dependencies {
  implementation("io.vertx:vertx-web-client:$vertxVersion")
  implementation("io.vertx:vertx-auth-jwt:$vertxVersion")
  implementation("io.vertx:vertx-auth-oauth2:$vertxVersion")
  implementation("io.vertx:vertx-unit:$vertxVersion")
  implementation("io.vertx:vertx-web:$vertxVersion")
  implementation("io.vertx:vertx-config:$vertxVersion")
  implementation("io.vertx:vertx-config-yaml:$vertxVersion")
  implementation("io.vertx:vertx-pg-client:$vertxVersion")
  implementation("io.vertx:vertx-lang-kotlin-coroutines:$vertxVersion")
  implementation("io.vertx:vertx-consul-client:$vertxVersion")
  implementation("io.vertx:vertx-lang-kotlin:$vertxVersion")
  // 日志
  implementation("org.apache.logging.log4j:log4j-slf4j18-impl:$log4j2Version")

  testImplementation("io.vertx:vertx-junit5:$vertxVersion")
  testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:$junitJupiterEngineVersion")
  testImplementation("org.junit.jupiter:junit-jupiter-api:$junitJupiterEngineVersion")
}
```

compileKotlin/compileTestKotlin

这里需要引入一个类，同时使用委托。官网强制要求这样写的

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile

// ......

val compileKotlin: KotlinCompile by tasks

compileKotlin.kotlinOptions {
  jvmTarget = "1.8"
}
val compileTestKotlin: KotlinCompile by tasks
```

```
compileTestKotlin.kotlinOptions {
  jvmTarget = "1.8"
}
```

shadowJar

这个就是一个 task 而已：

```
tasks.shadowJar {
  archiveClassifier.set("fat")
  manifest {
    // 需要使用 map
    attributes(mapOf("Main-Verticle" to mainVerticleName ))
  }
  mergeServiceFiles {
    include("META-INF/services/io.vertx.core.spi.VerticleFactory")
  }
}
```

test

这里也是一个 task，需要引入一下 Test 的枚举：

```
import org.gradle.api.tasks.testing.logging.TestLogEvent.*

// ......

tasks.withType<Test> {
  useJUnitPlatform()
  testLogging {
    events = mutableSetOf(PASSED, FAILED, SKIPPED)
  }
}
```

run

最后就是 run 了，这里我们换一个 task 的写法：

```
tasks {
  // 前面的 def 变量我拿到这里了
  val watchForChange = "src/**/*"
  val doOnChange = "./gradlew classes"
  // JavaExec 类型 task
  "run"(JavaExec::class) {
    // 运行参数
    args("run", mainVerticleName,
      "--redeploy=$watchForChange",
      "--launcher-class=${application.mainClassName}",
      "--on-redeploy=$doOnChange")
  }
}
```

最终结果

最后我们得到的 build.gradle.kts 如下：

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile
import org.gradle.api.tasks.testing.logging.TestLogEvent.*

plugins {
  application
  kotlin("jvm") version "1.3.20"
  id("com.github.johnrengelman.shadow") version "5.0.0"
}

group = "cn.edu.gzmu.center"
version = "1.0.0-SNAPSHOT"

repositories {
  maven("http://maven.aliyun.com/nexus/content/groups/public/")
  maven("https://oss.sonatype.org/content/repositories/snapshots")
  mavenCentral()
  jcenter()
}


application {
  mainClassName = "io.vertx.core.Launcher"
}
val mainVerticleName = "cn.edu.gzmu.center.MainVerticle"

val kotlinVersion by extra { "1.3.20" }
val vertxVersion by extra { "4.0.0-SNAPSHOT" }
val log4j2Version by extra { "2.13.0" }
val junitJupiterEngineVersion by extra { "5.4.0" }

dependencies {
  implementation("io.vertx:vertx-web-client:$vertxVersion")
  implementation("io.vertx:vertx-auth-jwt:$vertxVersion")
  implementation("io.vertx:vertx-auth-oauth2:$vertxVersion")
  implementation("io.vertx:vertx-unit:$vertxVersion")
  implementation("io.vertx:vertx-web:$vertxVersion")
  implementation("io.vertx:vertx-config:$vertxVersion")
  implementation("io.vertx:vertx-config-yaml:$vertxVersion")
  implementation("io.vertx:vertx-pg-client:$vertxVersion")
  implementation("io.vertx:vertx-lang-kotlin-coroutines:$vertxVersion")
  implementation("io.vertx:vertx-consul-client:$vertxVersion")
  implementation("io.vertx:vertx-lang-kotlin:$vertxVersion")
  implementation("org.apache.logging.log4j:log4j-slf4j18-impl:$log4j2Version")

  testImplementation("io.vertx:vertx-junit5:$vertxVersion")
  testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:$junitJupiterEngineVersion")
  testImplementation("org.junit.jupiter:junit-jupiter-api:$junitJupiterEngineVersion")
}

val compileKotlin: KotlinCompile by tasks
```

```
compileKotlin.kotlinOptions {
  jvmTarget = "1.8"
}
val compileTestKotlin: KotlinCompile by tasks

compileTestKotlin.kotlinOptions {
  jvmTarget = "1.8"
}


tasks.shadowJar {
  archiveClassifier.set("fat")
  manifest {
    attributes(mapOf("Main-Verticle" to mainVerticleName ))
  }
  mergeServiceFiles {
    include("META-INF/services/io.vertx.core.spi.VerticleFactory")
  }
}

tasks.withType<Test> {
  useJUnitPlatform()
  testLogging {
    events = mutableSetOf(PASSED, FAILED, SKIPPED)
  }
}

tasks {
  val watchForChange = "src/**/*"
  val doOnChange = "./gradlew classes"
  "run"(JavaExec::class) {
    args("run", mainVerticleName,
      "--redeploy=$watchForChange",
      "--launcher-class=${application.mainClassName}",
      "--on-redeploy=$doOnChange")
  }
}
```

IDEA 中 reimport，一下，然后等待加载。运行原来的 MainVerticle 和 Gradle 的 shadow 命令试能不能正常运行和打包。

整合 vertx-plugin

可以看到，我们的整个文件比较冗长，有差不多 90 行的配置。并且很多地方特别难看，比如版本号里。

在 Spring boot 的 Gradle 项目里，我们所有的 Spring 相关的依赖都交给了 io.spring.dependency management 插件来管理。那么 vert.x 应该也有这么一个东西，也就是 vertx-plugin。

我们接下来将它整合到项目中去。

plugins

因为 vertx-plugin 已经为我们整合了多款插件，包括 application 、shadow，所以我们删除他们

```
plugins {
  kotlin("jvm") version "1.3.20"
  id("io.vertx.vertx-plugin") version "1.0.1"
}
```

简单变化

- repositories：不变
- application：已经内置，直接移除
- extra：
  - val vertxVersion by extra { "4.0.0-SNAPSHOT" } 移除，交给插件管理。
  - val junitJupiterEngineVersion by extra { "5.4.0" } 移除，交给插件管理。
- shadowJar：移除，交给插件管理
- run：移除，交给插件管理

dependencies

移除版本号与单元测试依赖，插件自动添加

```
dependencies {
  implementation("io.vertx:vertx-web-client")
  implementation("io.vertx:vertx-auth-jwt")
  implementation("io.vertx:vertx-auth-oauth2")
  implementation("io.vertx:vertx-unit")
  implementation("io.vertx:vertx-web")
  implementation("io.vertx:vertx-config")
  implementation("io.vertx:vertx-config-yaml")
  implementation("io.vertx:vertx-pg-client")
  implementation("io.vertx:vertx-lang-kotlin-coroutines")
  implementation("io.vertx:vertx-consul-client")
  implementation("io.vertx:vertx-lang-kotlin")
  implementation("org.apache.logging.log4j:log4j-slf4j18-impl:$log4j2Version")
  testImplementation("io.vertx:vertx-junit5")
}
```

新增

新增两个，一个是指定插件运行环境（可以省略），另外一个是插件配置

```
tasks.withType<KotlinCompile> {
  kotlinOptions {
    freeCompilerArgs = listOf("-Xjsr305=strict")
    jvmTarget = "1.8"
  }
}

vertx {
```

```
  // 运行的主要 Verticle
  mainVerticle = "cn.edu.gzmu.center.WebVerticle"
  // 运行的版本号，默认 3.8.3
  vertxVersion = "4.0.0-SNAPSHOT"
}
```

最终结果

最后我们得到新的 build.gradle.kts

```
import org.jetbrains.kotlin.gradle.tasks.KotlinCompile
import org.gradle.api.tasks.testing.logging.TestLogEvent.*

plugins {
  kotlin("jvm") version "1.3.20"
  id("io.vertx.vertx-plugin") version "1.0.1"
}

group = "cn.edu.gzmu.center"
version = "1.0.0-SNAPSHOT"

repositories {
  maven("http://maven.aliyun.com/nexus/content/groups/public/")
  maven("https://oss.sonatype.org/content/repositories/snapshots")
  mavenCentral()
  jcenter()
}

val mainVerticleName = "cn.edu.gzmu.center.MainVerticle"
val kotlinVersion by extra { "1.3.20" }
val log4j2Version by extra { "2.13.0" }

dependencies {
  implementation("io.vertx:vertx-web-client")
  implementation("io.vertx:vertx-auth-jwt")
  implementation("io.vertx:vertx-auth-oauth2")
  implementation("io.vertx:vertx-unit")
  implementation("io.vertx:vertx-web")
  implementation("io.vertx:vertx-config")
  implementation("io.vertx:vertx-config-yaml")
  implementation("io.vertx:vertx-pg-client")
  implementation("io.vertx:vertx-lang-kotlin-coroutines")
  implementation("io.vertx:vertx-consul-client")
  implementation("io.vertx:vertx-lang-kotlin")
  implementation("org.apache.logging.log4j:log4j-slf4j18-impl:$log4j2Version")
  testImplementation("io.vertx:vertx-junit5")
}

val compileKotlin: KotlinCompile by tasks
compileKotlin.kotlinOptions {
  jvmTarget = "1.8"
}

val compileTestKotlin: KotlinCompile by tasks
```

```
compileTestKotlin.kotlinOptions {
  jvmTarget = "1.8"
}

tasks.withType<Test> {
  useJUnitPlatform()
  testLogging {
    events = mutableSetOf(PASSED, FAILED, SKIPPED)
  }
}

tasks.withType<KotlinCompile> {
  kotlinOptions {
    freeCompilerArgs = listOf("-Xjsr305=strict")
    jvmTarget = "1.8"
  }
}

vertx {
  mainVerticle = "cn.edu.gzmu.center.verticle.WebVerticle"
  vertxVersion = "4.0.0-SNAPSHOT"
}
```

最终只有 65 行代码。同时版本交给了插件来进行管理，方便省心。并且默认就配置了热部署和热 debug。

自动添加了如下命令：

- run
- runShadow
- startShadowScripts
- vertxDebug
- vertxRun

我们可以直接使用 vertx 提供的 vertxRun 和 vertxDebug ，运行 ./gradlew vertxRun 即可热运行实时更新。很棒！

整合 vertx-boot

在我们整合了 vertx-plugin 以后，已经很方便了。但是还是需要我们提供一个 mainVerticle，然后在 ainVerticle 里面配置我们要启动的 Verticle，还要在代码中配置其参数。所以是较为麻烦的，那么我如何改变呢？这里就用到了 vertx-boot 这个库。

他最重要的作用就是提供了一个配置文件让我们来配置我们要启动的 Verticle，可以怼他进行高级配，比如 Verticle 最大的特性就是直接可以集群部署。那么我们就可以直接指定他集群部署的实例数目再比如我们可以指定它的启动配置等等。还支持高级配置，例如将它配置成 worker verticle。

所以接下来我们来引入这个库，只需要两步就可以完成：

添加依赖

```
val vertxBootVersion by extra { "1.1.2" }

// ......

implementation("io.github.jponge:vertx-boot:$vertxBootVersion")
```

添加配置文件

在 resource 下添加 application.conf

```
vertx-boot {
  verticles {
    web {
      name = "cn.edu.gzmu.center.verticle.WebVerticle"
      instance = 3 // 创建 3 个实例
    }
  }
}
```

运行

我们修改 build.gradle.kts 的 vertx 函数：

```
vertx {
  // 必须为这个
  mainVerticle = "io.github.jponge.vertx.boot.BootVerticle"
  vertxVersion = "4.0.0-SNAPSHOT"
}
```

然后运行 ./gradlew vertxRun 即可看到效果！

IDEA 运行

如果我们希望使用 IDEA 运行，也非常简单！创建一个 main 函数即可

```
fun main() {
  Vertx.vertx().deployVerticle("io.github.jponge.vertx.boot.BootVerticle")
  // 或
  // Vertx.vertx().deployVerticle(BootVerticle::class.java.name)
}
```

然后运行这个 main 函数即可！

其他

初始化的项目默认使用的 Kotlin 是 1.3.20，我们可以将它改为最新的 1.3.61。同时使用的是 JDK11，他编译后的字节码是 11 的（**注：这个特性只有在1.3.30 以后版本中可以使用。**）

```
val compileKotlin: KotlinCompile by tasks
compileKotlin.kotlinOptions {
  // Target version of the generated JVM bytecode (1.6, 1.8, 9, 10, 11 or 12), default is 1.6
  jvmTarget = "11"
```

```
}

val compileTestKotlin: KotlinCompile by tasks
compileTestKotlin.kotlinOptions {
  // Target version of the generated JVM bytecode (1.6, 1.8, 9, 10, 11 or 12), default is 1.6
  jvmTarget = "11"
}
```

总结

vert.x 的国内生态真的不怎么样，用的人凤毛麟角，资料更是少之又少。但是他确实足够轻量，足够。在他上面可以实现很多东西，相比于 Spring 的开发模式，vert.x 更加锻炼一个人、更加考验一个。并且在多个方面都让自己思维上有不少的进步。这两款插件国内都没有任何介绍，都是自己不断翻档，一步一步的摸索找到的，然后啃文档弄上去。其中比较复杂的其实是 Groovy DSL 转 Koltin DSL 的过程，大多资料用的都是 Groovy，所以学习起来还是有点吃力的。不过 vert.x 真的棒，配合 Koltin 协程简直爽翻了！