

ForgeO

src_resources

Copyright (C) 2023 src_resources. All rights reserved.

Table of contents

1. ForgeGradle 文档 (中文翻译)	3
1.1 添加该插件	3
2. 向这篇文档做出贡献	4
2.1 格式指南	4
3. 入门	5
3.1 入门ForgeGradle	5
4. 配置选项	7
4.1 ForgeGradle配置	7
4.2 运行配置	10
4.3 进阶配置	13
5. 依赖	14
5.1 依赖	14
5.2 Jar-in-Jar	16
6. ForgeGradle 5 -> 6 迁移入门	18
6.1 添加Foojay Toolchains插件	18
6.2 将Gradle升级到8.1.1	18
6.3 更新ForgeGradle插件版本	18
6.4 从运行配置中移除 <code>forceExit</code>	19

1. ForgeGradle 文档 (中文翻译)

这里是ForgeGradle的官方文档，用于开发MinecraftForge以及使用MinecraftForge的模组的一个Gradle插件。

该文档 仅 针对ForgeGradle编纂，而不是一个Java、Groovy或Gradle教程。

如果你愿意对文档做出贡献，请阅读[向文档做出贡献](#)。

1.1 添加该插件

ForgeGradle使用Gradle 8；它可以使用 `build.gradle` 中的 `plugins` 块添加，方法是将以下信息添加到 `settings.gradle`：

```
// 在settings.gradle中
pluginManagement {
    repositories {
        // ...

        // 添加MinecraftForge maven
        maven { url = 'https://maven.minecraftforge.net/' }
    }
}

plugins {
    // 添加toolchain resolver
    id 'org.gradle.toolchains.foojay-resolver-convention' version '0.5.0'
}
```

```
// 在build.gradle中
plugins {
    // 添加ForgeGradle插件
    id 'net.minecraftforge.gradle' version '[6.0,6.2)'

    // ...
}
```

2. 向这篇文档做出贡献

你可以通过在MinecraftForge的官方[GitHub](#)仓库提交PR来提交对这篇文档的贡献；若要对翻译文档提出修改，请访问[中文翻译GitHub仓库](#)。

本文档的宗旨是明理达意。因此在提交贡献时，请解释它的工作原理，并将其拆分为合理的部分。我们在其他地方有一个[wiki](#)，可以展示更全面的代码示例（未指明具体地点——译者注）。

我们的受众是任何一位想要了解如何使用ForgeGradle的人。

请不要试图将这篇文档转变为一篇关于Java或Groovy开发的教程——后者是为那些了解类和其他基本结构如何工作的人准备的。

ForgeGradle的配置代码示例应使用Groovy语言编写。

2.1 格式指南

重要

请使用两个空格符来进行缩进，而不是制表符。

标题应该以标准的标题样式对首字母进行大写或小写。例如，

- **Guide For Contributing to This Documentation**（中文：本文档贡献指南）
- **Building and Testing Your Mod**（中文：构建并测试你的模组）

从本质上讲，除了不重要的单词外，所有单词都要大写。

拼写、语法和句法应遵循美式英语。此外，优先考虑使用用空格分割开的短语而不是缩写（例如“are not”而不是“aren't”）。

请使用等号和短划线，而不要使用“#”和“##”。对于h3及以下字体大小，“###”等也可以。此文件的源文件包含一个等号和短划线的示例。等号下划线创建h1文本，短划线下划线创建h2文本。

当引用代码块片段之外的字段和方法时，它们应该使用 # 分隔符（例如 `ClassName#methodName`）。内部类应该使用 \$ 分隔符（例如 `ClassName$InnerClassName`）。

JSON代码块片段应该使用 `js` 语法高亮显示。

所有链接都应在页面底部指定其URL。任何本文档的内部链接都应该通过其相对路径引用对应的页面。

警告（用 `!!! <type>` 表示）必须按照[该文档](#)所示进行设置；否则，它们最终可能会渲染不正确

3. 入门

3.1 入门ForgeGradle

如果你以前从未使用过ForgeGradle，以下是设置开发环境所需的最低信息量。

先决条件

- Java开发包（JDK）的安装

Minecraft版本	Java开发包版本
1.12 - 1.16	JDK 8
1.17	JDK 16
1.18 - 1.19	JDK 17

- 熟练使用一款集成开发环境（IDE）。
 - 建议使用一款集成了Gradle功能的IDE。

3.1.1 安装ForgeGradle

1. 首先从MinecraftForge下载Modder Development Kit（MDK）的一个副本，并将zip解压到一个空目录中。
2. 在你选择的IDE中打开提取MDK的目录。如果你的IDE集成了Gradle，请将其作为Gradle项目导入。
3. 为你的模组自定义你的Gradle构建脚本：
 1. 将 `archivesBaseName` 设置为所需的mod id。此外，用该mod id替换所有出现的 `examplemod`。
 2. 将 `group` 更改为所需的程序包名称。请确保遵循现有的命名约定。
 3. 更改 `version` 编号以反映你的模组的当前版本。强烈建议使用Forge对语义版本控制的扩展。

重要

确保对mod id的任何更改都反映在mods.toml和模组主类中。有关更多信息，请参阅Forge文档上的规划你的模组结构。

1. 使用IDE重新加载或刷新Gradle项目。如果你的IDE没有集成Gradle，请从项目目录中的shell运行以下命令：

```
./gradlew build --refresh-dependencies
```

1. 如果你的IDE是Eclipse、IntelliJ IDEA或Visual Studio Code，则可以分别使用以下命令之一生成运行配置：

Eclipse

```
./gradlew genEclipseRuns
```

IntelliJ IDEA

```
./gradlew genIntelliJRuns
```

Visual Studio Code

```
./gradlew genVSCodeRuns
```

你可以使用生成的运行配置之一来运行客户端、服务端等。

提示

如果你的IDE没有被列出，你仍然可以使用 `./gradlew run*`（例如，`runClient`、`runServer`、`runData`）。你也可以将这些命令与支持IDE一起使用。

恭喜你，现在你已经建立了一个开发环境！

4. 配置选项

4.1 ForgeGradle配置

ForgeGradle有许多配置可以改变开发环境的配置方式。大多数配置都是使用 `minecraft` 块设置的；然而，其他一些可以在 `dependencies` 块中指定，或者修改构建的 `jar`，例如 `reobfJar`。

4.1.1 启用访问转换器

访问转换器可以扩大可见性或修改Minecraft类、方法和字段的 `final` 标志。要在生产环境中启用访问转换器，可以将 `accessTransformer` 设置为有问题的配置文件：

```
minecraft {
    // ...

    // 添加相对于项目目录的访问转换器文件
    accessTransformer = project.file('src/main/resources/META-INF/accesstransformer.cfg')
}
```

重要

虽然开发环境中的访问转换器可以从用户指定的任何位置读取，但在生产中，该文件只能从 `META-INF/accesstransformer.cfg` 读取。

4.1.2 人类可读的映射

Minecraft的源代码被混淆了。因此，所有类、方法和字段都具有机器生成的名称，而没有包结构。同时，由于本地变量表的存储方式，函数的局部变量名变成了雪人(☹)。使用模糊名称创建模组很困难，因为反向工程很乏味，可能会在不同版本之间更改，并且虽然可能在语言本身中无效，但在字节码中却不是无效的。

为了解决最后两个问题，Forge通过**ForgeAutoRenamingTool**将每个类、方法、字段和参数模糊地映射到一个唯一的标识符，即SRG名称。当游戏由用户客户端运行时，SRG映射在生产中使用。

为了便于开发，ForgeGradle允许用户选择一个映射集来应用于SRG映射之上，我们将其称为人类可读的映射。ForgeGradle知道如何通过 `reobf*` 任务将模组jar转换为SRG映射，以便在生产中使用。

可以通过设置 `minecraft` 块中的 `mappings` 字段来指定所使用的映射集。`mappings` 字段接受两个参数：`channel` 是映射集的类型，`version` 是要应用的映射集的版本。

目前，ForgeGradle中内置了三个默认映射集：

- `official` - 这使用Mojang提供的映射集。这些映射没有参数名称，仅从1.14开始存在。
- `stable` - 这使用MCP生成的映射集。这些通常是不完整的，并且自1.17起不再存在。
- `snapshot` - 这也是MCP生成的映射集，类似于程序的隔夜构建。这些通常也是不完整的，并且从1.17起不再存在。

注意

生产中使用的类名从1.17之前的 `stable` 到1.17之后的 `official` 。

```
mappings {
    // 将映射设置为在Minecraft 1.19.4中使用来自Mojang的映射。
    mappings channel: 'official', version: '1.19.4'

    // ...
}
```

Parchment

Parchment是ParchmentMC维护的一个官方项目，它在 `official` 映射集之上提供开放的、社区源代码的参数名称和 javadoc。你可以在[他们的网站](#)上学习如何设置和使用Parchment映射。

4.1.3 准备运行任务

运行任务（`run*`）有两个独立的管道，这取决于它们是通过 `gradlew` 还是运行配置执行的。默认情况下，有两个任务用于准备工作区以便执行：

首先，有在 `run*` 任务之前执行的 `prepare*` 任务，并确保为游戏准备映射文件。`prepare*Compile` 任务通常仅作为 `run*` 任务的依赖项来执行，以确保游戏在运行之前已编译。

如果你的IDE是Eclipse或IntelliJ IDEA，则可以将运行配置配置为在启动游戏之前执行 `prepare*` 任务，方法是分别将 `enableEclipsePrepareRuns` 或 `enableIdeaPrepareRuns` 设置为 `true`。这将允许你在IDE启动游戏之前调用自定义Gradle任务。

```
minecraft {
    // ...

    // 为运行配置启用'prepare*'任务
    enableEclipsePrepareRuns true
    enableIdeaPrepareRuns true
}
```

复制IDE资源

`copyIdeResources` 属性可用于将 `processResources` 任务配置的资源复制到IDE的资源输出目录。这允许不调用Gradle（IntelliJ配置为使用IDEA运行器或Eclipse）的IDE运行配置使用构建脚本可配置资源。通常，在替换 `mods.toml` 等文件中的值时，需要启用此属性。这仅通过 `copyEclipseResources` 和 `copyIntelliJResources` 任务分别适用于Eclipse和IntelliJ IDEA。

```
minecraft {
    // ...

    // 将文件从'processResources'复制到IDE的资源输出目录
    copyIdeResources true
}
```

运行配置文件夹

如果 `generateRunFolders` 设置为 `true`，则可以将运行配置排序到文件夹中。这将读取特定[运行配置][`run`]中设置的 `folderName` 属性，以确定组织性的结构。

```
minecraft {
    // ...
```

```
// 如果为true, 运行配置将按其 'folderName' 分组到文件夹中  
generateRunFolders true  
}
```

4.2 运行配置

运行配置定义了游戏实例的运行方式。这包括参数、工作目录、任务名称等。运行配置在 `minecraft.runs` 块中定义。虽然默认情况下没有配置任何运行，但 **Forge** 确实提供了 `client`、`server`、`data` 或 `gameTestServer` 的配置。

```
minecraft {
  // ...
  runs {
    // 在此处配置运行
  }
}
```

可以使用闭包添加类似于任何 `NamedDomainObjectContainer` 的运行配置。

```
// 在minecraft块内部
runs {
  // 创建或配置名为'client'的运行配置
  client {
    // 配置运行
  }
}
```

以下配置属性可用：

```
// 在runs块内部
client {
  // Gradle运行任务的名称，
  // 默认为'runX'，其中X是容器名称
  taskName 'runThing'

  // 设置程序启动的入口点
  // Forge将userdev main设置为'cpw.mods.bootstraplauncher.BootstrapLauncher'
  main 'com.example.Main'

  // 设置该配置的工作目录
  // 默认为 './run'
  workingDirectory 'run'

  // 为IntelliJ IDEA设置要为其运行配置的模块的名称
  // 默认为 '<project_name>.main'
  ideaModule 'example.main'

  // 设置运行配置应添加到的文件夹的名称
  // 默认为项目名称
  folderName 'example'

  // 设置这是否应该运行一个Minecraft客户端
  // 若未指定，将进行以下检查
  // - 是否存在包含'client'的环境变量'thing'
  // - 配置名称是否存在'client'
  // - main是否设置为'mcp.client.Start'
  // - main是否设置为'net.minecraft.client.main.Main'
  client true
}
```

```

// 设置该配置应继承自的父级
parent runs.example

// 设置该配置的子级
children runs.child

// 合并此配置并指定是否覆盖现有属性
merge runs.server, true

// 如果不为false, 则会将父级的参数与此配置合并
inheritArgs false

// 如果不为false, 则会将父级的JVM参数与此配置合并
inheritJvmArgs false

// 将一个源集 (sourceset) 添加到classpath
// 若未指定, 则添加sourceSet.main
source sourceSets.api

// 为该运行设置一个环境变量
// 值将作为一个文件 (file) 或一个字符串 (string) 被解释
environment 'envKey', 'value'

// 设置一个系统属性
// 值将作为一个文件 (file) 或一个字符串 (string) 被解释
property 'propKey', 'value'

// 设置将传递给应用的参数
// 可用'args'指定多个
arg 'hello'

// 设置一个JVM参数
// 可用'jvmArgs'指定多个
jvmArg '-Xmx2G'

// 设置一个令牌 (token)
// 目前, 下列令牌被使用:
// - runtime_classpath
// - minecraft_classpath
token 'tokenKey', 'value'

// 设置一个被惰性初始化的令牌
// 通常应替代'token'使用, 例如当令牌解析Gradle配置时
lazyToken('lazyTokenKey') {
    'value'
}

// 如果为true, 则编译所有项目, 而不是当前任务的项目
// 这仅由IntelliJ IDEA使用
buildAllProjects false
}

```

提示

你可以在[MinecraftForge构建脚本](#)中看到所有配置的userdev属性的列表。

4.2.1 模组配置

当前环境中的模组可以使用运行配置中的 `mods` 块添加。Mod块也是 `NamedDomainObjectContainer`。

```
// 在runs块中
client {
    // ...

    mods {
        other_mod {
            // ...
        }

        // 配置'example'模组
        example {
            // 将一个源集添加到模组的源
            source sourceSets.main

            // 合并此配置并指定是否覆盖现有属性
            merge mods.other_mod, true
        }
    }
}
```

4.3 进阶配置

ForgeGradle包含一些特定或细微差别的配置技术，具体取决于你的构建项目的复杂性。

4.3.1 重新混淆源集

默认情况下，`reobf* abd rename*` 任务只包含主源集classpath上的文件。要在不同的classpath上重新生成文件，需要将它们添加到任务中的 `libraries` 属性中。

```
// 添加另一个源集的classpath到'reobf'任务。
tasks.withType('reobfJar') {
    libraries.from sourceSets.api.classpath
}
```

5. 依赖

5.1 依赖

依赖关系不仅用于开发模组之间的互操作性或为游戏添加额外的库，还决定了为哪个版本的Minecraft开发。这将提供关于如何修改 `repositories` 和 `dependencies` 块以将依赖项添加到开发环境的快速概述。

这不会深入解释Gradle的概念。强烈建议你在继续之前阅读[Gradle依赖管理指南](#)。

5.1.1 minecraft

`minecraft` 依赖项指定要使用的Minecraft版本，并且必须包含在 `dependencies` 块中。除了具有组 `net.minecraft` 的artifact外，任何artifact都将应用随依赖项提供的任何补丁。这通常只指定 `net.minecraftforge:forge` artifact。

```
dependencies {
    // Forge artifact的版本遵循格式'<mc_version>-<forge_version>'
    // 'mc_version' 是要加载的Minecraft的版本（例如，1.19.4）
    // 'forge_version'对于那个Minecraft版本想要的Forge的版本（例如，45.0.23）
    // 原版可用'net.minecraft:joined:<mc_version>'编译作为替代
    minecraft 'net.minecraftforge:forge:1.19.4-45.0.23'
}
```

5.1.2 模组依赖

在一个典型的开发环境中，Minecraft被降级为中间映射，用于生产，然后转换为模组指定的任何人类可读的映射。模组artifact在构建时会被混淆为生产映射（SRG），因此无法直接用作Gradle依赖项。

因此，在添加到预期配置之前，所有模组依赖项都需要用 `fg.deobf` 包装。

```
dependencies {
    // 假设我们已经指定了'minecraft'依赖

    // 假设我们有某个可从一个指定仓库获取的artifact 'examplemod'
    implementation fg.deobf('com.example:examplemod:1.0')
}
```

本地模组依赖

如果你试图依赖的模组在Maven存储库上不可用（例如，[Maven Central](#)、[CurseMaven](#)、[Modrinth](#)），你可以使用[平坦目录][flat directory]添加模组依赖：

```
repositories {
    // 将项目目录中的'libs'文件夹添加为一个平坦目录
    flatDir {
        dir 'libs'
    }
}

dependencies {
```

```
// ...

// 给定某个<group>:<name>:<version>:<classifier (default None)>
// 具有指定的扩展名<ext (默认为jar)>
// 平坦目录中的项目将按以下顺序解决：
// - <name>-<version>.<ext>
// - <name>-<version>-<classifier>.<ext>
// - <name>.<ext>
// - <name>-<classifier>.<ext>

// 如果明确指定了一个分类器 (classifier)
// 具有分类器的artifact将具有优先级：
// - examplemod-1.0-api.jar
// - examplemod-api.jar
// - examplemod-1.0.jar
// - examplemod.jar
implementation fg.deobf('com.example:examplemod:1.0:api')
}
```

注意

组名称可以是任何内容，但对于平坦目录条目，不能为空，因为解析artifact文件时不会检查这些条目。

5.1.3 非Minecraft依赖

在开发环境中，Forge默认不会加载非Minecraft依赖项。为了让Forge识别非Minecraft依赖项，必须将它们添加到`minecraftLibrary`配置中。`minecraftLibrary`的工作原理类似于Gradle中的`implementation`配置，在编译时和运行时应用。

```
dependencies {
    // ...

    // 假设有一个非Minecraft库'dummy-lib'
    minecraftLibrary 'com.dummy:dummy-lib:1.0'
}
```

默认情况下，添加到开发环境中的非Minecraft依赖项不会包含在构建的artifact中！你必须使用Jar-In-Jar在构建时将依赖项包含在artifact中。

5.2 Jar-in-Jar

Jar-in-Jar是从模组的jar中加载模组依赖项的一种方式。为了实现这一点，Jar-in-Jar在构建时在

`META-INF/jarjar/metadata.json` 中生成一个元数据json，其中包含要从jar中加载的artifact。

Jar-in-Jar是一个完全可选的系统，可以在 `minecraft` 块之前使用 `jarJar#enable` 来启用。这将把 `jarJar` 配置中的所有依赖项都包含到 `jarJar` 任务中。你能够以类似于其他jar任务的方式来配置该任务：

```
// 在build.gradle中

// 为你的模组启用Jar-in-Jar系统
jarJar.enable()

// 配置'jarJar'任务
// 'all'是默认的分类器 (classifier)
tasks.named('jarJar') {
    // ...
}
```

5.2.1 添加依赖

你可以使用 `jarJar` 配置添加要包含在jar中的依赖项。由于Jar-in-Jar是一个协商（negotiation）系统，所有版本都应该提供一个支持的范围。

```
// 在build.gradle中
dependencies {
    // 根据exampleLib在从2.0（包括）到3.0（不包括）之间的最高支持版本编译并包含该版本
    jarJar(group: 'com.example', name: 'exampleLib', version: '[2.0,3.0)')
}
```

如果需要指定要包含的确切版本，而不是该范围中支持的最高版本，则可以在依赖关系闭包中使用 `jarJar#pin`。在这些情况下，`artifact`版本将在编译时使用，而固定（pinned）版本将捆绑在模组jar中。

```
// 在build.gradle中
dependencies {
    // 根据exampleLib在从2.0（包括）到3.0（不包括）之间的最高支持版本编译并包含该版本
    jarJar(group: 'com.example', name: 'exampleLib', version: '[2.0,3.0)') {
        // 包含exampleLib 2.8.0
        jarJar.pin(it, '2.8.0')
    }
}
```

你可以在针对特定版本进行编译时附加固定一个版本范围：

```
// 在build.gradle中
dependencies {
    // 编译exampleLib 2.8.0
    jarJar(group: 'com.example', name: 'exampleLib', version: '2.8.0') {
        // 包含从2.0（包括）到3.0（不包括）之间的最高支持版本的exampleLib
        jarJar.pin(it, '[2.0,3.0)')
    }
}
```

```
}
}
```

使用运行时依赖

如果你想在jar中包含模组的运行时依赖项，你可以在构建脚本中调用 `jarJar#fromRuntimeConfiguration`。如果你决定使用此选项，强烈建议你包含依赖项筛选器；否则，包括Minecraft和Forge在内的每一个依赖项都将捆绑在jar中。为了支持更灵活的语句，`dependency` 配置已添加到 `jarJar` 扩展和任务中。使用此选项，可以指定要包含在配置中或从配置中排除的模式：

```
// 在build.gradle中

// 为jar添加运行时依赖项
jarJar.fromRuntimeConfiguration()

// ...

jarJar {
    // 在此处包含或排除运行时依赖项
    dependencies {
        // 排除以'com.google.gson.'开头的任何依赖项
        exclude(dependency('com.google.gson.*'))
    }
}
```

提示

使用 `#fromRuntimeConfiguration` 时，通常建议至少设置一个 `include` 筛选器。

5.2.2 将一个Jar-in-Jar发布到Maven

出于存档原因，ForgeGradle支持将Jar-in-Jar artifact发布给选定的Maven，类似于Shadow插件处理它的方式。在实践中，这不是有用的，也不推荐使用。

```
// I在build.gradle中（具有'maven-publish'插件）

publications {
    mavenJava(MavenPublication) {
        // 添加标准java组件和Jar-in-Jar artifact
        from components.java
        jarJar.component(it)

        // ...
    }
}
```

6. ForgeGradle 5 -> 6 迁移入门

这是一个关于如何将你的构建脚本从ForgeGradle 5迁移到6的高级、非详尽的概述。

如果有任何不正确或遗漏的信息，请在下面留言。谢谢！

6.1 添加Foojay Toolchains插件

Gradle现在使用Foojay Toolchains插件以管理Java工具链。必须将插件添加到 `settings.gradle[.kts]` 中：

使用Gradle DSL：

```
// 在settings.gradle中
plugins {
    id 'org.gradle.toolchains.foojay-resolver-convention' version '0.5.0'
}
```

使用Kotlin DSL：

```
// 在settings.gradle.kts中
plugins {
    id("org.gradle.toolchains.foojay-resolver-convention") version "0.5.0"
}
```

6.2 将Gradle升级到8.1.1

首先，你需要更新Gradle Wrapper以使用8.1.1。你可以在[Gradle的迁移指南](#)上找到有关从7迁移到8的其他信息。

这可以通过以下两种方式之一实现：

6.2.1 gradle wrapper

首选选项是将 `gradle wrapper` 命令与 `--gradle-version` 选项一起使用。

```
# 在Gradle项目的根目录内部
./gradlew wrapper --gradle-version=8.1.1
```

6.2.2 gradle-wrapper.properties

不建议使用此方法，并且只有在上面的 `gradlew wrapper` 命令失败时才应使用此方法。

更新 `gradle/wrapper/gradle-wrapper.properties` 中的 `distributionUrl` 属性中的版本。然后运行 `./gradlew wrapper`。

```
distributionUrl=https\://services.gradle.org/distributions/gradle-8.1.1-bin.zip
```

6.3 更新ForgeGradle插件版本

更新Gradle后，将 `build.gradle[.kts]` 中的 `ForgeGradle` 版本更新为 `[6.0.6.2]`：

使用Gradle DSL:

```
// 在build.gradle中
plugins {
    id 'net.minecraftforge.gradle' version '[6.0,6.2)'
}
```

使用Kotlin DSL:

```
// 在build.gradle.kts中
plugins {
    id("net.minecraftforge.gradle") version "[6.0,6.2)"
}
```

6.4 从运行配置中移除 forceExit

ForgeGradle现已删除运行配置中的 `forceExit` 属性。对运行任务的更改仅适用于当前项目，不适用于任何子项目。

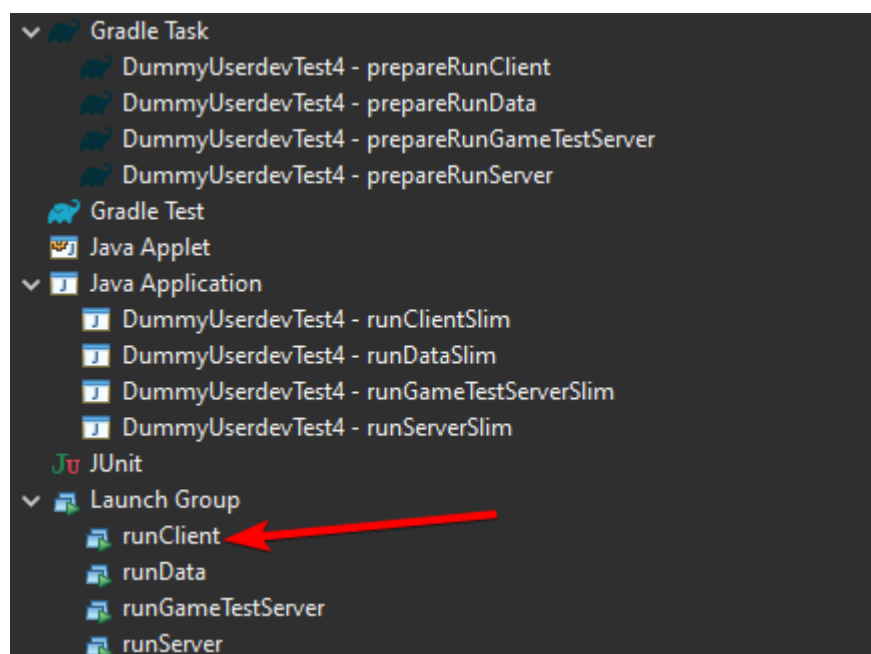
6.4.1 次要的添加、变更和移除

6.4.2 重新混淆的Classpath继承

如果你在 `reobf*` 任务配置中使用 `classpath.from` 来声明额外的库，则需要考虑 `reobf*` 任务；它应该迁移到 `libraries.from` 属性。

6.4.3 Eclipse启动器组 (Launcher Groups)

如果ForgeGradle的设置使得Eclipse应该在开始游戏之前运行Gradle任务，那么你必须从 `Launch Group` 文件夹执行运行配置。只有在 `minecraft` 块中启用了新属性时，情况才会如此。



6.4.4 ModConfig中 resources 和 classes 的移除

运行配置的 `mods` 块中的属性 `resources` 和 `classes` 已从ForgeGradle 6中移除。