1. 背景

自从开始使用 Android Studio 以来，Gradle 已成为管理依赖项、配置数据、风味等许多事情的中心位置。随着时间的推移，维护跨多模块项目的依赖项已成为一项挑战。让我们详细了解跨多模块项目的依赖管理演变是如何发生的以及面临的问题。

让我们用一个多模块的应用程序来理解这个问题。手动声明版本以及跨多个模块的依赖项是我们的第一种方法。

当我们在每个 build.gradle 文件中定义依赖版本时，如果没有正确升级，就会产生版本冲突等问题。管理依赖和配置数据的版本升级非常困难。每当我们有更多模块时，依赖关系发生变化时，就会有很多手动过程去检查每个模块并检查依赖关系是否退出，如果发现更新然后移动到下一个。

1、配置项目范围的属性

然后我们开始使用 Gradle Extra 属性设置项目范围的属性。对于这种方法，我们习惯于在项目或根级 build.gradle 文件中定义配置数据和依赖版本





这解决了升级依赖项版本的问题，将它们保持在一个中心位置。但是，没有用于检查版本或自动建议的导航支持。

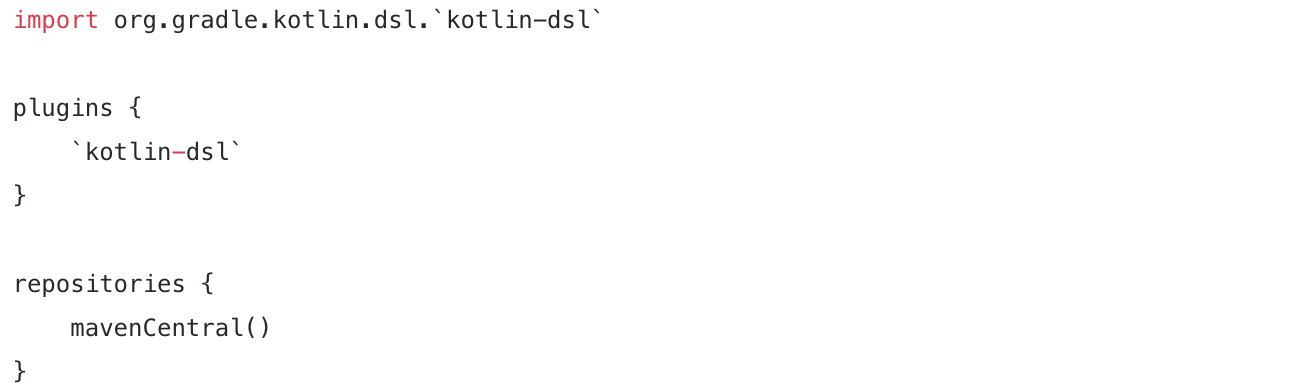
2、什么是 buildSrc？

buildSrc 是项目根级别的目录，其中包含构建信息。我们可以使用这个目录来启用 kotlin-dsl 并编写与自定义配置相关的逻辑并在整个项目中共享它们。

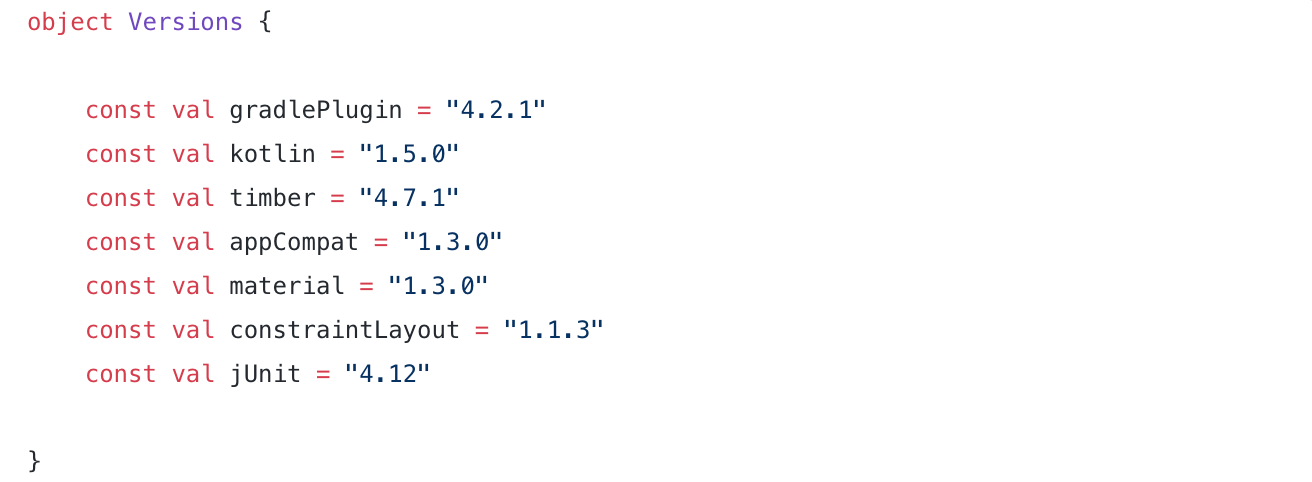
目录 buildSrc 被视为包含的构建。发现目录后，Gradle 会自动编译和测试此代码，并将其放入构建脚本的类路径中。对于多项目构建，只能有一个 buildSrc 目录，它必须位于项目根目录中。 buildSrc 应该优先于脚本插件，因为它更易于维护、重构和测试代码。

在 buildSrc 中启用 Kotlin DSL

由于 Kotlin DSL 是从父 Kotlin 语言中采用的，因此它的大部分语法都是相似的。为此，首先在 buildSrc 中创建一个空文件 build.gradle.kts，然后在 buildSrc 中启用“kotlin-dsl”选项



在 buildSrc 目录中创建一个 src目录结构,让我们创建一个名为 Versions.kt 的对象类，我们在其中定义与插件、依赖项等相关的所有版本.

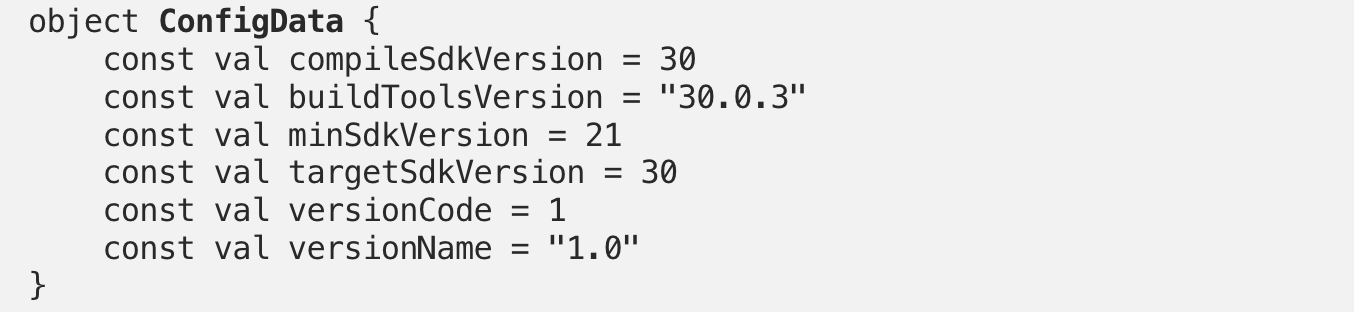


让我们创建一个新文件 Dependencies.kt 来定义我们所有的插件、依赖项等



这样子就可以通过Dependencies的配置来查看Versions的具体数值。

我们还可以为任何构建配置相关信息创建一个 ConfigData.kt



最后在模块级build.gradle.kts文件中使用上述类中定义的数据



buildSrc + Kotlin DSL 是依赖管理的最佳选择。因为它是一个类级别的声明，所以可以很容易地测试。自动建议支持和代码导航将有助于节省时间。为每个目的维护单独的类。这种方法可以很容易地实现更好的可重用性和易于维护。

1. Version catalogs

在项目之间共享依赖项

Gradle 用户最常提出的问题之一是如何在项目之间正确共享依赖版本。例如，假设您有一个使用此布局的多项目构建：



因为它们存在于同一个“多项目”中，所以预计客户端和服务器都需要相同的依赖项。



如果没有任何共享机制，两个项目都会复制依赖声明，这会带来一些缺点：

- 升级库需要更新所有使用它的构建文件

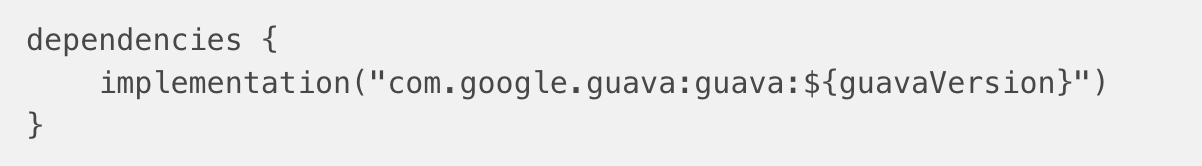
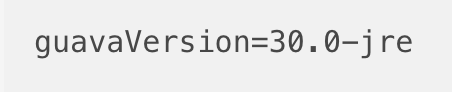
- 必须记住所有依赖项的依赖项坐标（组、工件、版本）

- 可能会不小心在不同的项目中使用不同的版本

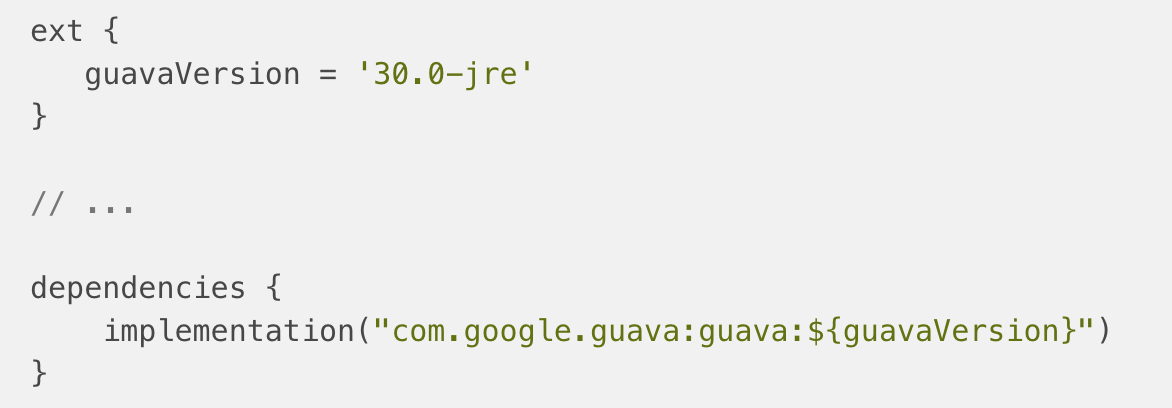
- 一些依赖项总是一起使用，但您必须在构建文件中复制条目

有三种方式可以共享：

- gradle.properties配置：

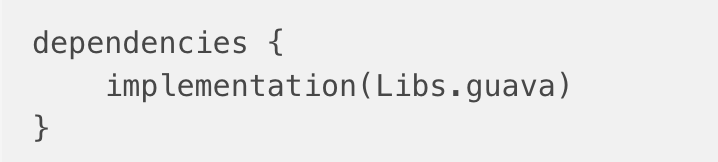


- extra properties配置：



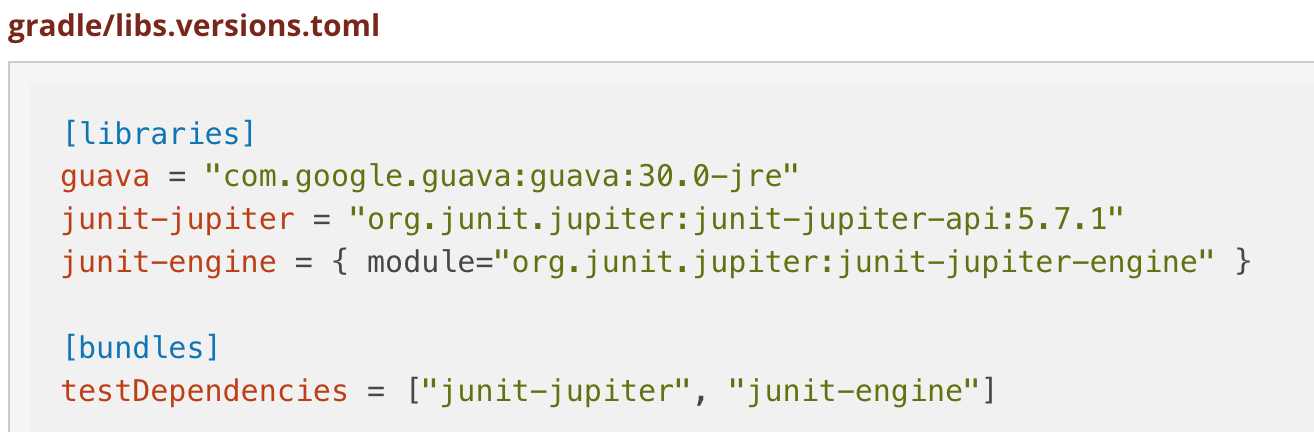
- 在 buildSrc 中声明库，然后使用类型安全访问器在构建脚本中声明依赖项：



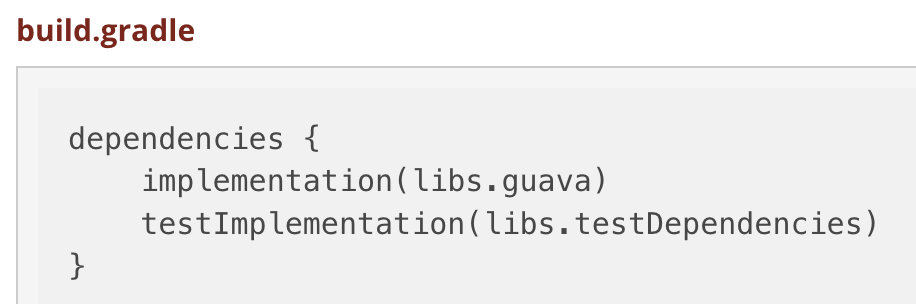


buildSrc 方案的缺点是：对任何依赖项的任何更改都将触发构建脚本的重新编译并使构建脚本类路径无效，从而导致最新的检查失败，并最终重建比您应该为单个版本更改所做的更多的事情。

版本目录基本上是 Gradle 支持的所有以前模式的替代品，没有以前方法的缺点。目录是在常规位置找到的文件，使用 TOML 配置格式：



这声明了将在构建脚本中使用的依赖坐标。您仍然必须声明您的依赖项，但现在可以使用类型安全 API 来完成：



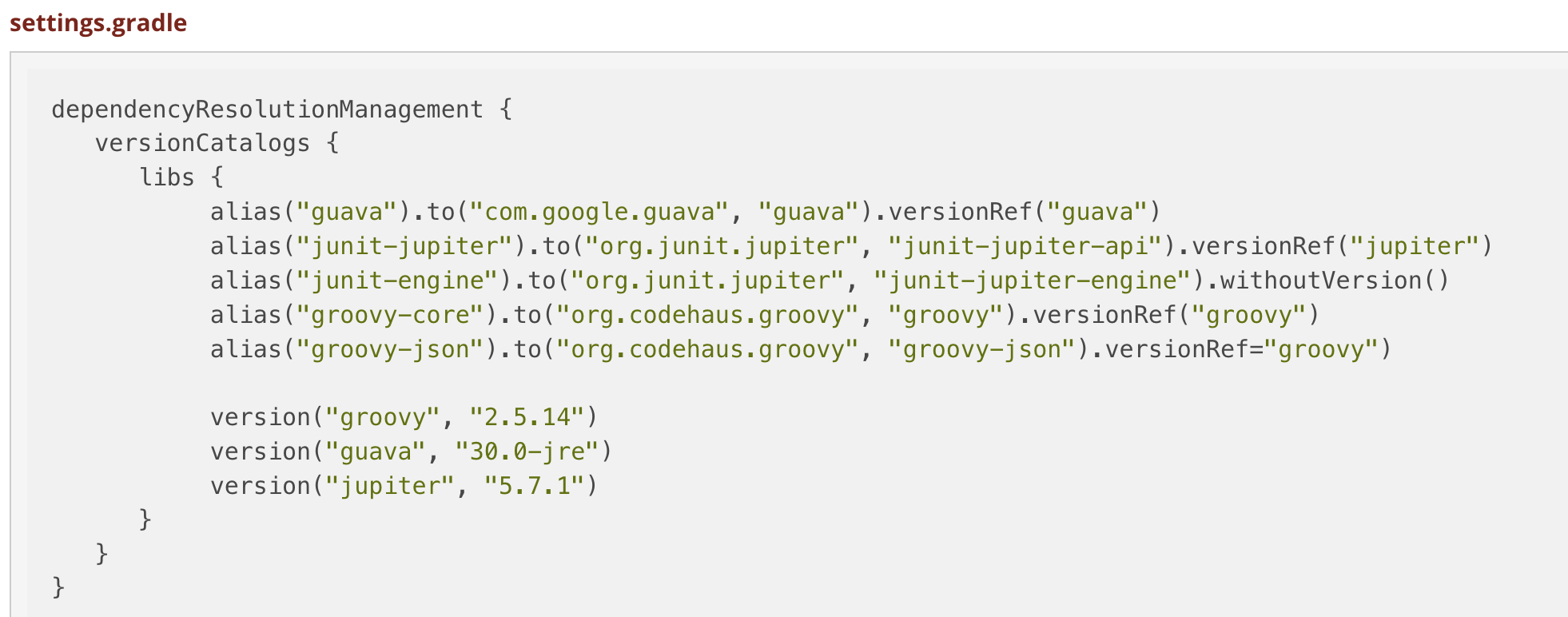
在上面的目录文件中，我们直接在坐标中内联了依赖版本。但是，可以将它们外部化，以便您可以在依赖项之间共享依赖项版本。例如：



这带来了其他好处，例如更新依赖项的 GAV 坐标（组、工件或版本）不会触发重新编译构建脚本。 TOML 格式还为我们提供了声明丰富版本的能力。

在底层，Gradle 提供了一个 API 来声明目录。此 API 位于设置中，这意味着插件作者可以贡献目录，例如通过应用于 settings.gradle(.kts) 文件的约定插件。

此 API 比使用 TOML 文件时更冗长，但专为类型安全而设计。上面目录的等价物是这样的：



如果您正在使用外部目录，则实际上必须使用此 API。这是该功能的一大卖点：它允许团队（或框架作者）发布目录，以便用户获得推荐。例如，假设 Spring Boot 团队发布了一个推荐目录（他们今天对 BOM 做了类似的事情，但 BOM 会对你可能不想要的传递依赖产生影响）。

在 Gradle 构建中使用此目录将如下所示：



这将使您的构建脚本中的 spring 命名空间下的目录可用。因此，您可以通过声明此依赖项来使用 Spring 团队推荐的任何 SLF4J 版本：



这样的目录将作为 TOML 文件发布在常规 Maven 存储库中。由于 Gradle 的高级依赖解析引擎，对于用户来说，真正的依赖是一个目录是完全透明的。