

HenCoder Plus 第 21 课 讲义

Gradle Plugin

Groovy 两个语法点

- getter / setter
 - 每个 field, Groovy 会自动创建它的 getter 和 setter 方法, 从外部可以直接调用, 并且在使用 `object.fieldA` 来获取值或者使用 `object.fieldA = newValue` 来赋值的时候, 实际上会自动转而调用 `object.getFieldA()` 和 `object.setFieldA(newValue)`
- 字符中的单双引号
 - 单引号是不带转义的, 而双引号内的内容可以使用 `"string1${var}string2"` 的方式来转义

Gradle Plugin

- 本质: 把逻辑独立的代码抽取和封装

Plugin 的最基本写法

写在 `build.gradle` 里:

`build.gradle:`

```
class PluginDemo implements Plugin<Project> {

    @Override
    void apply(Project target) {
        println 'Hello author!'
    }
}

apply plugin: PluginDemo
```

Extension

`build.gradle:`

```
class ExtensionDemo {
    def author = 'Kai'
}

class PluginDemo implements Plugin<Project> {

    @Override
```

```

void apply(Project target) {
    def extension = target.extensions.create('hencoder', ExtensionDemo)
    target.afterEvaluate {
        println "Hello ${extension.author}!"
    }
}

}

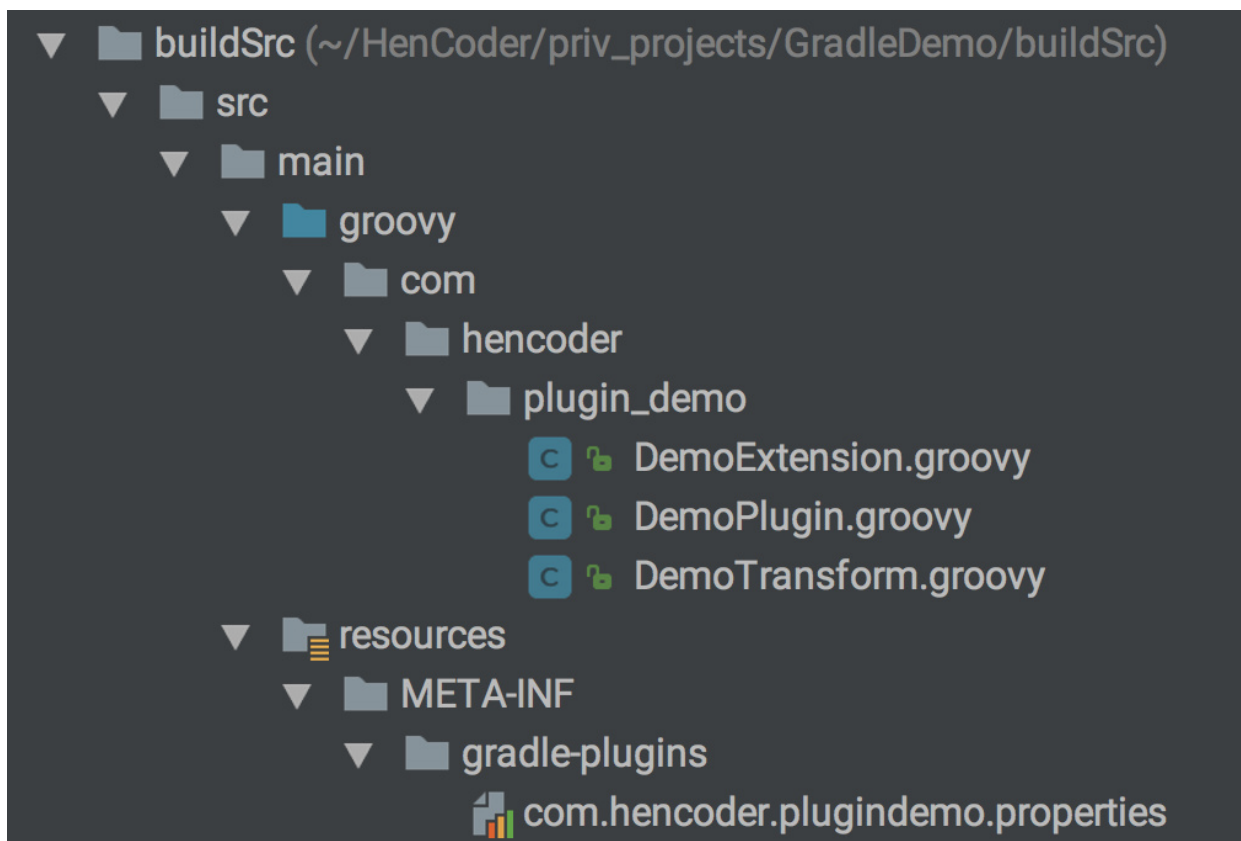
apply plugin: PluginDemo

hencoder {
    author 'rengwuxian'
}

```

写在 buildSrc 目录下

- 目录结构:



- `resources/META-INF/gradle-plugins/*.properties` 中的 `*` 是插件的名称, 例如 `*.properties` 是 `com.hencoder.plugindemo.properties`, 最终在应用插件是的代码就应该是:

```

apply plugin: 'com.hencoder.plugindemo'

```

- `*.properties` 中只有一行, 格式是:

```

implementation-class=com.hencoder.plugin_demo.DemoPlugin

```

其中等号右边指定了 Plugin 具体是哪个类

- Plugin 和 Extension 写法和在 `build.gradle` 里的写法一样
- 关于 buildSrc 目录
 - 这是 gradle 的一个特殊目录，这个目录的 build.gradle 会自动被执行，即使不配置进 settings.gradle
 - buildSrc 的执行早于任何一个 project，也早于 settings.gradle。它是一个独立的存在
 - buildSrc 所配置出来的 Plugin 会被自动添加到编译过程中的每一个 project 的 classpath，因此它们才可以直接使用 `apply plugin: 'xxx'` 的方式来便捷应用这些 plugin
 - settings.gradle 中如果配置了 `':buildSrc'`，buildSrc 目录就会被当做是子 Project，因此会被执行两遍。所以在 `settings.gradle` 里面应该删掉 `':buildSrc'` 的配置

Transform

- 是什么：是由 Android 提供了，在项目构建过程中把编译后的文件（jar 文件和 class 文件）添加自定义的中间处理过程的工具
- 怎么写
 - 先加上依赖：

```
// 因为 buildSrc 早于任何一个 project 执行，因此需要自己添加仓库
repositories {
    google()
    jcenter()
}

dependencies {
    implementation 'com.android.tools.build:gradle:3.1.4'
}
```

- 然后继承 `com.android.build.api.transform.Transform`，创建一个子类：

```
class DemoTransform extends Transform {

    // 构造方法
    DemoTransform() {
    }

    // 对应的 task 名
    @Override
    String getName() {
        return 'hencoderTransform'
    }

    // 你要对那些类型的结果进行转换(是字节码还是资源文件?)
    @Override
    Set<QualifiedContent.ContentType> getInputTypes() {
        return TransformManager.CONTENT_CLASS
    }
}
```

```

    }

    // 适用范围包括什么(整个 project 还是别的什么?)
    @Override
    Set<? super QualifiedContent.Scope> getScopes() {
        return TransformManager.SCOPE_FULL_PROJECT
    }

    @Override
    boolean isIncremental() {
        return false
    }

    // 具体的「转换」过程
    @Override
    void transform(TransformInvocation transformInvocation) throws
    TransformException, InterruptedException, IOException {
        def inputs = transformInvocation.inputs
        def outputProvider = transformInvocation.outputProvider

        inputs.each {
            // jarInputs: 各个依赖所编译成的 jar 文件
            it.jarInputs.each {
                // dest:
                //
                ../app/build/intermediates/transforms/hencoderTransform/...
                File dest =
outputProvider.getContentLocation(it.name, it.contentTypes,
it.scopes, Format.JAR)
                FileUtils.copyFile(it.file, dest)
            }

            // directoryInputs: 本地 project 编译成的多个 class 文件存
            放的目录
            it.directoryInputs.each {
                // dest:
                //
                ../app/build/intermediates/transforms/hencoderTransform/...
                File dest =
outputProvider.getContentLocation(it.name, it.contentTypes,
it.scopes, Format.DIRECTORY)
                FileUtils.copyDirectory(it.file, dest)
            }
        }
    }
}

```

- 还能做什么：修改字节码 上面的这段代码只是把编译完的内容原封不动搬运到目标位置，没有实际用处。要修改字节码，需要引入其他工具，例如 javassist。javassist 的使用教程在网上有很多，可以搜索一下。