码炫课堂基础篇之-gradle快速入门

码炫课堂版权所有
交流Q群：963060292

码炫课堂技术交流群：963060292

群名称:码炫课堂java架构群1
群　号:963060292

码炫课堂版权所有 交流群：963060292

讲师简介

smart哥，互联网悍将，历经从传统软件公司到大型互联网公司的洗礼，入行即在中兴通讯等大型通信公司担任项目leader，后随着互联网的崛起，先后在美团支付等大型互联网公司担任架构师，公派旅美期间曾与并发包大神Doug Lea探讨java多线程等最底层的核心技术。对互联网架构底层技术有相当的研究和独特的见解，在多个领域有着丰富的实战经验。

一、前言

Ant和Maven共享在Java市场上相当大的成功。ANT是在2000年发布了第一个版本的工具，它是基于程序编程思想的发展。 后来，人们在 Apache-Ivy的帮助下，网络接受插件和依赖管理的能力有所提升。但主要缺点是使用XML作为一种格式来写构建脚本。XML是分层的，不利于程序的编程，而且当XML文件变大以后变得难以管理。 Maven在2004年推出的，它比ANT有一个很大的改进。它改变了结构并且继续使用XML编写生成规范。Maven的依赖约定和能够通过网络下载依赖关系。Maven的主要好处是它的生命周期。虽然接连的多个项目生命周期相同，这是以灵活性为代价的。 Maven也面临着依赖管理的一些问题。它不会在同一库版本之间处理好矛盾，复杂的定制构建脚本实际上Maven比ANT更难写。 最后，Gradle于2012年发布，带来了一些更高效的特点。 Gradle是一个基于Apache Ant和Apache Maven概念的项目自动化建构工具。它使用一种基于Groovy的特定领域语言(DSL)来声明项目设置，抛弃了基于XML的各种繁琐配置。 面向Java应用为主。当前其支持的语言限于Java、Groovy和Scala，计划未来将支持更多的语言。

https://www.bilibili.com/video/BV1ST4y177Tx

2000 ant---------ivy

2004 maven----resipotory

2012 gradle

java ，groovy，scala

xx.class

jvm

码炫课堂版权所有
交流Q群：963060292

1、gradle特性

- 按约定声明构建和建设；
- 强大的支持多工程的构建；
- 强大的依赖管理（基于Apache Ivy），提供最大的便利去构建工程；
- 全力支持已有的 Maven 或者Ivy仓库基础建设；
- 支持传递性依赖管理，在不需要远程仓库和pom.xml和ivy配置文件的前提下；
- 基于groovy脚本构建，其build脚本使用groovy语言编写；
- 具有广泛的领域模型支持构建；
- 深度 API；
- 易迁移；
- 自由和开放源码，Gradle是一个开源项目，基于 ASL 许可。

2、为什么使用Groovy?

完整的Gradle API是使用Groovy语言设计的。这是基于XML内部 DSL 的优点。Gradle是其核心的通用构建工具; 它的主要焦点是Java项目。在这些项目中，团队成员要熟悉Java，这是为了更好的构建透明，给所有团队成员的项目。

类似于 Python，Groovy或Ruby语言是最好的构建框架。为什么Groovy被选中？这是因为它为使用Java的人提供了迄今为止最大的透明度。Groovy的基本语法与Java是一样的。

二、Gradle安装配置

在本文档中，我们将学习 Gradle 的安装，对于一个初学者，有时安装开发环境也是一个比较麻烦的问题。

如果按照 Gradle 官方网站的说明安装，则可能会遇到一些麻烦，有时还要在互联网上做一些搜索，查找为什么进入命令提示符输入 gradle -v 得不到任何东西。

下面是完整的步骤来安装 Gradle ，为我们后续的 Gradle学习的开发环境作好准备！

1、前提条件

windows8 及以上

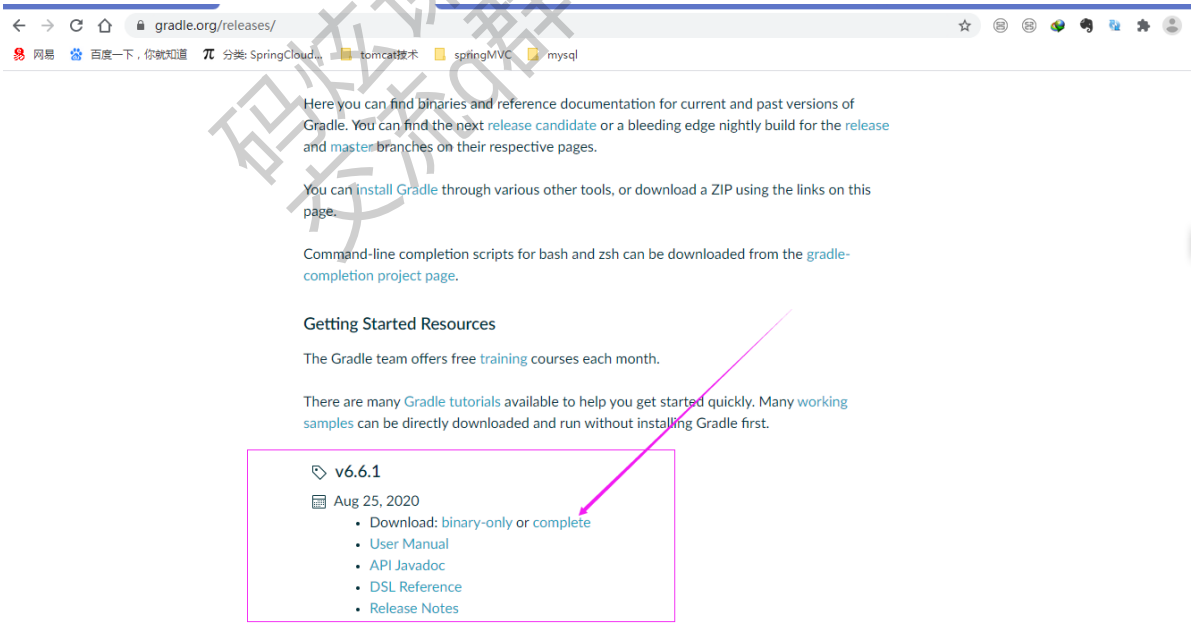
jdk8及以上

```
C:\Users\ling>java -version
java version "1.8.0_40"
Java(TM) SE Runtime Environment (build 1.8.0_40-b25)
Java HotSpot(TM) 64-Bit Server VM (build 25.40-b25, mixed mode)
```

2、安装说明

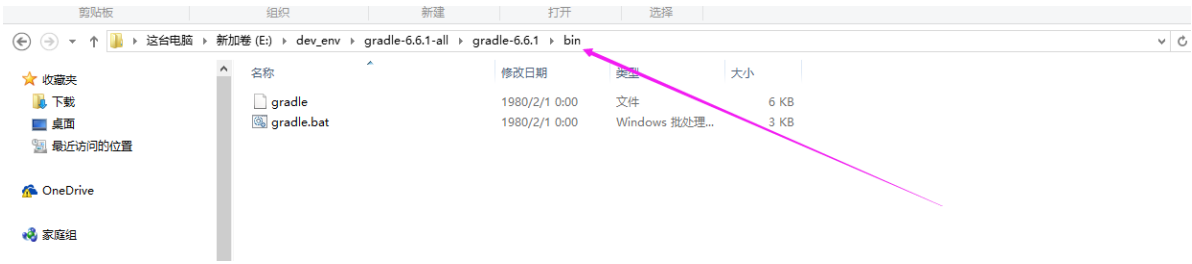
官网下载：https://gradle.org/

下载最新6.6.1完全版本（包含源码）

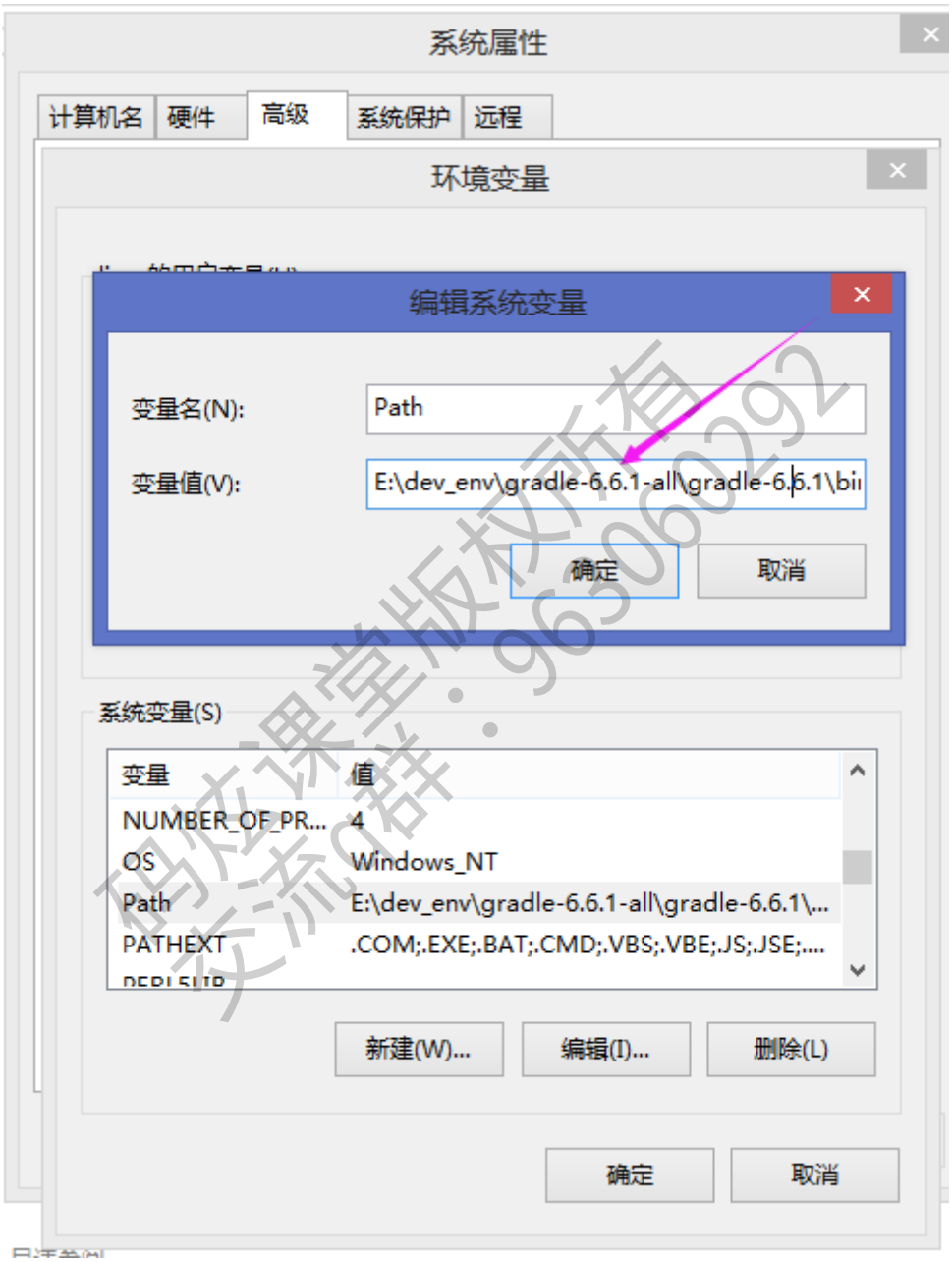


下载解压后配置环境变量

码悠课堂版权所有 交流Q群：963060292



到bin这一层



配置完在cmd窗口执行：gradle -v

```
C:\Users\ling>gradle -v

Welcome to Gradle 6.6.1!

Here are the highlights of this release:
 - Experimental build configuration caching
 - Built-in conventions for handling credentials
```

```
 - Java compilation supports --release flag

For more details see https://docs.gradle.org/6.6.1/release-notes.html


------------------------------------------------------------
Gradle 6.6.1
------------------------------------------------------------

Build time:   2020-08-25 16:29:12 UTC
Revision:     f2d1fb54a951d8b11d25748e4711bec8d128d7e3

Kotlin:       1.3.72
Groovy:       2.5.12
Ant:          Apache Ant(TM) version 1.10.8 compiled on May 10 2020
JVM:          1.8.0_40 (Oracle Corporation 25.40-b25)
OS:           Windows 8.1 6.3 amd64
```

三、Gradle构建脚本

Gradle构建脚本文件用来处理两件事情：一个是项目和另一个就是任务。

每个`Gradle`生成表示一个或多个项目。一个项目表示一个JAR库或Web应用程序，也可能表示由其他项目产生的JAR文件组装的ZIP。

简单地说，一个项目是由不同的任务组成。一个任务是指构建执行的一块工作。任务可能是编译一些类，创建一个`JAR`，产生的`Javadoc`或发布一些归档文件库。

1、编写构建脚本

Gradle提供了一个域特定语言(DSL)，用于描述构建。它使用 Groovy 语言，使其更容易来形容和构建。Gradle 中的每一个构建脚本使用UTF-8进行编码保存，并命名为 `build.gradle` 。

创建build.gradle文件

演示脚本内容：它将打印 “this is the first gradle“。复制并保存以下脚本到文件：`D:\mypro\gvy`。

脚本中定义一个任务名称 `firstgvy`，这是用来打印”this is the first gradle“ 字符串。内容如下：

```
task firstgvy {
  doLast {
    println 'this is the first gradle'
  }
}
```

注：doLast是task的一个action

在命令提示符下，进入存储 `build.gradle` 文件的目录并执行以下命令，得到结果如下所示 -

```
D:\mypro\gvy>gradle -q firstgvy
this is the first gradle
```

码农课堂版权所有 交流QQ：953060292

Gradle如何使用Groovy

Gradle 构建脚本使用 Groovy API。作为一个入门，我们来来看看下面的例子。

以下示例演示将字符串转换为大写。

复制并保存下面的代码到 `build.gradle` 文件。

```
task upper {
  doLast {
    String expString = 'maxuan first gradle'
    println "Original: " + expString
    println "Upper case: " + expString.toUpperCase()
  }
}
```

使用 `gradle -q upper` 执行上面的代码，如果命令执行成功，会得到下面的输出。

```
D:\mypro\gvy>gradle -q upper
this is the first gradle
Original: maxuan first gradle
Upper case: MAXUAN FIRST GRADLE
```

下面的例子 4 次打印隐式参数 (`$it`) 的值。复制并保存下面的代码到 `build.gradle` 文件。

```
task count {
  doLast {
    4.times {
      print "$it "
    }
    println ""
  }
}
```

使用 `gradle -q count` 执行上面的代码，如果命令执行成功，会得到下面的输出。

```
D:\mypro\gvy>gradle -q count
0 1 2 3
```

码炫课堂版权所有
交流q群：963060292

2、build的三个阶段

gradle 构建的生命周期主要分为三个阶段，Initialization，Configuration，Execution。

- Initialization：Gradle支持单个或多个工程的构建。在Initialization阶段，Gradle决定哪些工程将参与到当前构建过程，并为每一个这样的工程创建一个Project实例。一般情况下，参与构建的工程信息将在settings.gradle中定义。
- Configuration：在这一阶段，配置project的实例。所有工程的构建脚本都将被执行。Task，configuration和许多其他的对象将被创建和配置。
- Execution：在之前的configuration阶段，task的一个子集被创建并配置。这些子集来自于作为参数传入gradle命令的task名字，在execution阶段，这一子集将被依次执行。

问题：doLast语句处于哪个阶段？

试验演示

3、Groovy的JDK方法

Groovy增加了很多有用的方法到标准的Java类。例如，从Java API可迭代实现它遍历Iterable接口的元素的`each()`方法。

复制并保存下面的代码到`build.gradle`文件。

```
task groovyJDKMethod {
   String myName = "maxuan";
   myName.each() {
      println "${it}"
   };
}
```

使用`gradle -q groovyJDKMethod`执行上面的代码，如果命令执行成功，会得到下面的输出。

```
D:\mypro\gvy>gradle -q  groovyJDKMethod
m
a
x
u
a
n
```

码炫课堂版权所有
交流Q群：963060292

四、Gradle任务

Gradle构建脚本描述一个或多个项目，每个项目都由不同的任务组成。

任务是构建执行的一项工作，可以是编译一些类，将类文件存储到单独的目标文件夹中，创建JAR，生成Javadoc或将一些归档发布到存储库。

1、定义任务

任务是用于将任务定义到构建脚本中的关键字。看看下面的例子，它是一个叫作maxuan的任务，将打印一个字符串：“码炫课堂”。将以下脚本复制并保存到`build.gradle`文件中。 此构建脚本定义一个名称为 “maxuan” 的任务，用于打印`码炫课堂`字符串。

```
task maxuan {
   doLast {
      println 'maxuan course'
   }
}
```

在存储`build.gradle`文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q maxuan
maxuan course
```

您可能已经猜到，可以声明依赖于其他任务的任务。下面声明依赖于其他任务的任务，将以下代码复制并保存到`build.gradle`文件中。

```
task maxuan {
    println 'maxuan course'
}

task maxuanNB(dependsOn: maxuan) {
    println "NB! "
}
```

在存储`build.gradle`文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q maxuanNB
maxuan course
NB!
```

task之间能否相互依赖？

将以下代码复制并保存到`build.gradle`文件中。

```
task taskX(dependsOn: taskY) {
    println 'taskX'
}
task taskY(dependsOn: taskX){
    println 'taskY'
}
```

结论：任务间不能相互依赖

在存储`build.gradle`文件的目录位置执行以下命令，报错如下：

```
D:\mypro\gvy>gradle -q  taskX

FAILURE: Build failed with an exception.

* Where:
Build file 'D:\mypro\gvy\build.gradle' line: 33

* What went wrong:
A problem occurred evaluating root project 'gvy'.
> Could not get unknown property 'taskY' for root project 'gvy' of type org.grad
le.api.Project.

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug
option to get more log output. Run with --scan to get full insights.

* Get more help at https://help.gradle.org

BUILD FAILED in 1s
```

码悦课堂版权所有 交流Q群：963060292

2、定位任务

如果要查找在构建文件中定义的任务，则必须使用相应的标准项目属性。这意味着每个任务都可以作为项目的属性，使用任务名称作为属性名称。
看看下面的代码访问任务作为属性。将以下代码复制并保存到`build.gradle`文件中。

```
task maxuan

println maxuan.name
println project.maxuan.name
```

在存储`build.gradle`文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q  maxuan
maxuan
maxuan
```

您还可以通过任务集合使用所有属性。
将以下代码复制并保存到`build.gradle`文件中。

```
task hello
task hello1

println tasks.hello.name
println tasks['hello'].name
println tasks['hello1'].name
```

在存储`build.gradle`文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q  hello
hello
hello
hello1
```

码农课堂版权所有
交流Q群：963060292

3、向任务添加依赖关系

要将一个任务依赖于另一个任务，这意味着当一个任务完成时，另一个任务将开始。 每个任务都使用任务名称进行区分。 任务名称集合由其任务集合引用。 要引用另一个项目中的任务，应该使用项目路径作为相应任务名称的前缀。
以下示例将从任务`taskX`添加依赖项到任务`taskY`。

```
task taskX {
   println 'taskX'
}
task taskY(dependsOn: taskX) {
   println "taskY"
}
```

在存储 `build.gradle` 文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q taskY
taskX
taskY
```

上面的例子是通过使用**名称**添加对任务的依赖， 还有另一种方法实现任务依赖性，即使用Task对象定义依赖性。
现在采用上面任务的相同示例，但是使用任务**对象**而不是任务**名称**来实现依懒关系。
将以下代码复制并保存到`build.gradle`文件中。

```
task taskY {
   doLast{
     println 'taskY'
   }
}
task taskX {
   doLast{
     println 'taskX'
   }
}
taskY.dependsOn taskX
```

在存储 `build.gradle` 文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q taskY
taskX
taskY
```

还有另一种方法来添加任务依赖，它就是通过使用闭包，在这种情况下，任务通过闭包释放如果您在构建脚本中使用闭包，那么应该返回任务对象的单个任务或集合。

以下示例将任务中从`taskX`添加依赖项到项目中的所有任务，其名称以“`lib`”开头。
将以下代码复制并保存到`build.gradle`文件中。

```
task taskX {
  doLast{
   println 'taskX'
  }
}

taskX.dependsOn {
   tasks.findAll {
     task -> task.name.startsWith('lib')
   }
}
task lib1 {
  doLast{
   println 'lib1'
  }
```

码炫课堂版权所有
大流q群：963060292

```
}
task lib2 {
  doLast{
   println 'lib2'
  }
}
task notALib {
  doLast{
   println 'notALib'
  }
}
```

在存储 `build.gradle` 文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q taskX
lib1
lib2
taskX
```

4、向任务添加描述

可以向任务添加描述。 执行`Gradle`任务时会显示此描述。 这可以通过使用`description`关键字。
将以下代码复制并保存到`build.gradle`文件中。

```
task copy(type: Copy) {
   description 'Copies the resource directory to the target directory.'
   from 'resources'
   into 'target'
   include('**/*.txt', '**/*.xml', '**/*.properties')
   println("description applied")
}
```

在存储 `build.gradle` 文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q copy
description applied
```

5、跳过任务

如果用于跳过任务的逻辑不能用谓词表示，则可以使用`StopExecutionException`。 如果操作抛出此异常，则会跳过此操作的进一步执行以及此任务的任何后续操作的执行, 构建继续执行下一个任务。
将以下代码复制并保存到`build.gradle`文件中。

```
task compile {
    doLast{
       println 'We are doing the compile.'
    }
}
```

码农课堂版权所有
大流群：963060292

```
compile.doFirst {
    // Here you would put arbitrary conditions in real life.
    // But this is used in an integration test so we want defined behavior.
    if (true) { throw new StopExecutionException() }
}
task myTask(dependsOn: 'compile') {
    doLast{
       println 'I am not affected'
    }
}
```

在存储 `build.gradle` 文件的目录位置执行以下命令，应该看到输出结果如下 -

```
D:\mypro\gvy>gradle -q myTask
I am not affected
```

Gradle在处理任务时有不同的阶段, 首先，有一个配置阶段，其中直接在任务的闭包中指定的代码被执行, 针对每个可用任务执行配置块，而不仅针对稍后实际执行的那些任务。

五、Gradle依赖管理

`Gradle` 构建脚本定义了构建项目的过程; 每个项目包含一些依赖项和一些发表项。依赖性意味着支持构建项目的东西，例如来自其他项目的所需 `JAR` 文件以及类路径中的外部 `JAR`（如 `JDBC JAR` 或 `Eh-cache JAR`）。发布表示项目的结果，如测试类文件和构建文件，如 `war` 文件。

`Gradle` 负责构建和发布结果。 发布基于定义的任务。 可能希望将文件复制到本地目录，或将其上传到远程Maven或Ivy存储库，或者可以在同一个多项目构建中使用另一个项目的文件。 发布的过程称为发布。

1、声明依赖关系

`Gradle` 遵循一些特殊语法来定义依赖关系。 以下脚本定义了两个依赖项，一个是 `Hibernate core 5.4.21`，第二个是 `Junit 5.0` 和更高版本。如下面的代码所示，可在 `build.gradle` 文件中使用此代码。

```
apply plugin: 'java'

repositories {
   mavenCentral()
}

dependencies {
   compile group: 'org.hibernate', name: 'hibernate-core', version:
'5.4.21.Final'
   testCompile group: 'junit', name: 'junit', version: '5.+'
}
```

对比maven配置

程序猿版权所有 交流q群：962060292

```
<dependencies>
   <dependency>
            <groupId>junit</groupId>
            <artifactId>junit</artifactId>
            <version>5.0</version>
            <scope>test</scope>
  </dependency>
</dependencies>
```

2、依赖关系配置

依赖关系配置只是定义了一组依赖关系。 您可以使用此功能声明从Web下载外部依赖关系。这定义了以下不同的标准配置。

- 编译 – 编译项目的生产源所需的依赖关系。
- 运行时 - 运行时生产类所需的依赖关系。 默认情况下，还包括编译时依赖项。
- 测试编译 - 编译项目测试源所需的依赖项。 默认情况下，它包括编译的产生的类和编译时的依赖。
- 测试运行时 - 运行测试所需的依赖关系。 默认情况下，它包括运行时和测试编译依赖项。

3、外部依赖

外部依赖是一种依赖。 这是对当前构建之外构建的一些文件的依赖，并且存储在某种类型的存储库中，例如：Maven central，corporate Maven或Ivy repository或本地文件系统中的目录。
以下代码片段是定义外部依赖关系。 在`build.gradle`文件中使用如下代码。

```
dependencies {
   compile group: 'org.hibernate', name: 'hibernate-core', version:
'5.4.21.Final'
}
```

4、存储库

在添加外部依赖关系时， Gradle在存储库中查找它们。 存储库只是文件的集合，按分组，名称和版本来组织构造。 默认情况下，Gradle不定义任何存储库。 我们必须至少明确地定义一个存储库。

一个Java工程通常会依赖于外部的jar包，Gradle可以使用Maven的仓库来获取或者发布相应的jar包。

Gradle配置Maven中央仓库

下面的代码片段定义了如何定义`maven`仓库。 在`build.gradle`文件中使用此代码。

```
repositories {
   mavenCentral()
}
```

下面的代码是定义远程`maven`。 在`build.gradle`文件中可使用下面代码。

码炫课堂版权所有
交流Q群：963502292

```
repositories {
   maven {
      url "http://repo.mycompany.com/maven2"
   }
}
```

5、发布文件

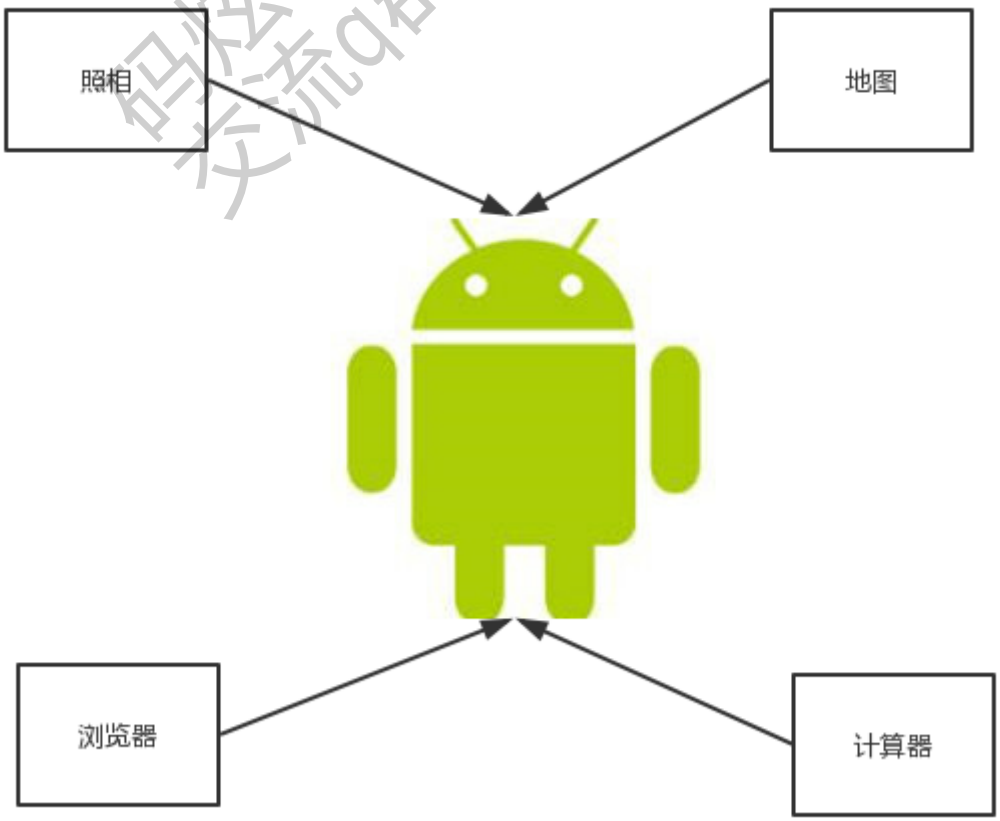
依赖关系配置也用于发布文件。 这些已发布的文件称为工件。 通常，我们使用插件来定义工件。 但是需要告诉 `Gradle` 在哪里发布文件。可以通过将存储库附加到上传存档任务来实现此目的。 请查看以下用于发布 `Maven` 存储库的语法。 执行时， `Gradle` 将根据项目需求生成并上传 `pom.xml` ， 在 `build.gradle` 文件中使用此代码。

```
apply plugin: 'maven'

uploadArchives {
   repositories {
      mavenDeployer {
         repository(url: "file://localhost/tmp/myRepo/")
      }
   }
}
```

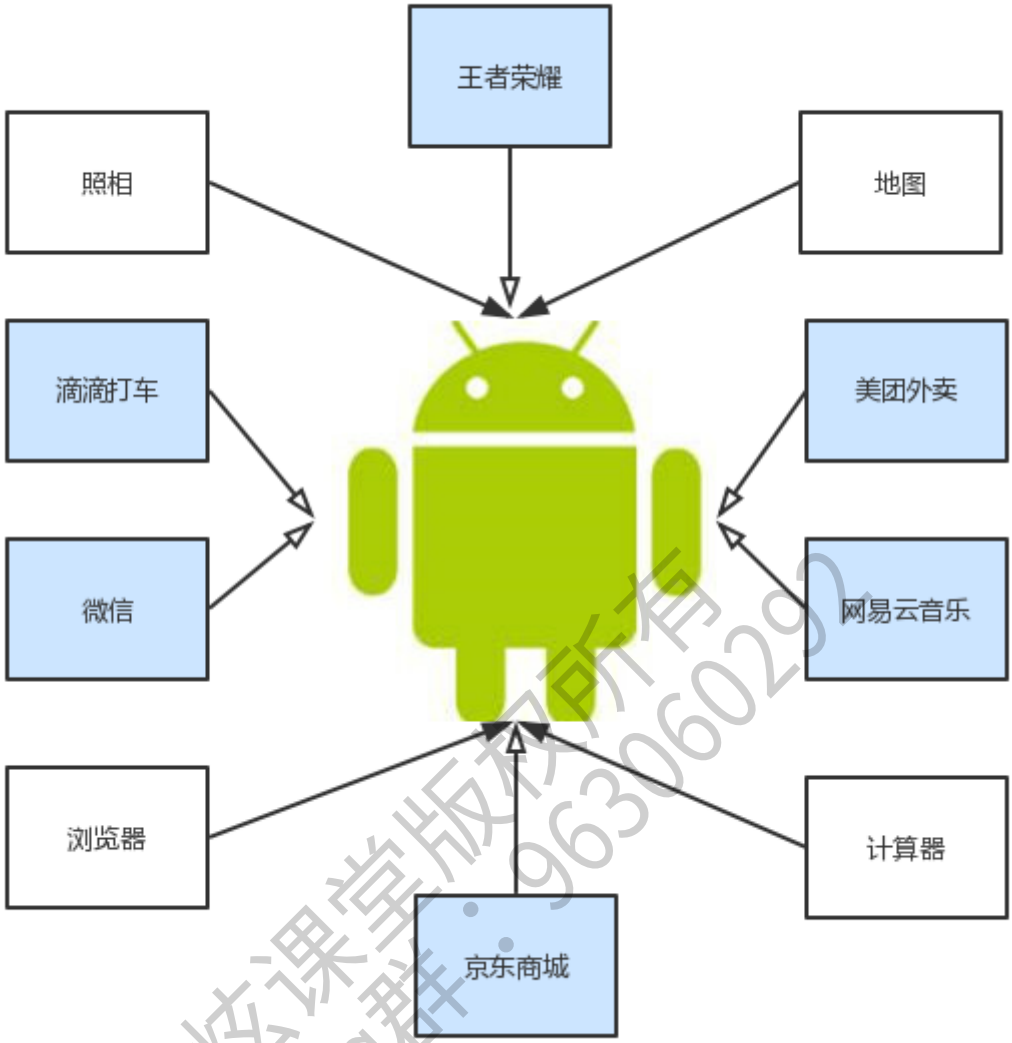
六、Gradle插件

插件只是一组任务，几乎所有的任务，如编译任务，设置域对象，设置源文件等都由插件处理。



码农课堂版权所有
交流q群：963060292

可以看到初始的机器人只有照相、地图、浏览器、计算机等功能，这显然是比较乏味的，我们可以给这个机器人安装很多其他的应用，使它提供更多的功能，如下图所示:



码炫课堂版权所有
交流Q群：963060292

我们给这个机器人安装了很多应用，这些应用不仅覆盖了人的衣食住行还提供了娱乐功能，我们可以玩游戏、听音乐和购物等等，机器人也得到了极大的提升，能够购为人类提供更多的服务。这些安装的应用可以理解为插件，这个插件可以自由的进行插拔，比如我们需要玩游戏时可以安装王者荣耀，如果不好玩就把它卸载掉。这么说来其实Android、iOS、Mac等操作系统采用的都是这种思想，而Gradle也是如此。
Gradle本身和初始的机器人一样，只是提供了基本的核心功能，其他的特性比如编译Java源码的能力等等就需要通过插件来实现了。

1、插件类型

Gradle中有两种类型的插件：**脚本插件**和**二进制插件**。

- 脚本插件是一个额外的构建脚本，它提供了一种声明性方法来操作构建，通常在构建中使用。
- 二进制插件是实现插件接口并采用编程方法来操作构建的类，二进制插件可以驻留在插件JAR中的一个构建脚本和项目层次结构或外部。

2、应用插件

Project.apply（ ）API方法用于应用特定的插件,可以多次使用相同的插件。

有两种类型的插件：一种是脚本插件，第二种是二进制插件。

1）脚本插件

脚本插件可以从本地文件系统进行应用，文件系统位置相对于项目目录，而远程脚本位置指定`HTTP URL`。

如下代码，它将`other.gradle`插件用于构建脚本， 在`build.gradle`文件中使用此代码如下。

新建other.gradle文件

```
ext{
    verson='1.0'
    url='http://maxuan.cn'
}
```

应用该脚本插件

```
apply from: 'other.gradle'
task test{
    doLast{
        println "version:$verson,url:$url"
    }
}
```

2）二进制插件（对象插件）

每个插件由插件标识，一些核心插件是使用短名称来应用它，一些社区插件是使用插件ID的完全限定名称，有时它允许指定一个插件类，**二进制插件就是实现了org.gradle.api.Plugin接口的插件**，它们可以有Plugin id。

通常对象插件分2类：

1、内部插件

以下代码片段， 它显示如何使用应用Java插件， 在`build.gradle`文件中使用方式如下：

```
apply plugin: JavaPlugin
```

以下代码，使用短名称应用核心插件， 在`build.gradle`文件中使用此代码如下所示。

```
plugins {
   id 'java'
}
```

以下代码，使用短名称应用社区（https://plugins.gradle.org/）插件。 在`build.gradle`文件中使用此代码如下所示。

尚硅谷课堂版权所有
交流Q群：963060292

```
plugins {
   id "com.maxuan.bintray" version "0.1.0"
}
```

2、第三方插件

第三方的对象插件通常是jar文件，要想让构建脚本知道第三方插件的存在，需要使用buildscrip来设置。

```
buildscript {
  repositories {
    maven {
      url "https://plugins.gradle.org/m2/"
    }
  }
  dependencies {
    classpath "com.jfrog.bintray.gradle:gradle-bintray-plugin:1.8.4"
  }
}
apply plugin: "com.jfrog.bintray"
```

3、编写自定义插件

在创建自定义插件时，需要编写一个插件的实现， `Gradle` 实例化插件并使用 `Plugin.apply()` 方法调用插件实例。

以下示例包含一个简单的 `hello` 插件，它将一个问候任务添加到项目中，在 `build.gradle` 文件中使用此代码。

```
apply plugin: HelloPlugin

class HelloPlugin implements Plugin<Project> {
   void apply(Project project) {
      project.task('hello') {
        doLast{
          println "Hello from the HelloPlugin."
        }
      }
   }
}
```

使用以下代码执行上述脚本。

```
D:\mypro\gvy> gradle -q hello
Hello from the HelloPlugin
```

码悟课堂版权所有 交流Q群：963060292

4、自定义插件扩展

大多数插件需要从构建脚本中的配置获得支持， `Gradle`项目有一个关联“`ExtensionContainer`”对象，它有助于跟踪传递给插件的所有设置和属性。

我们在项目中添加一个简单的扩展对象，例如添加一个问候语扩展对象，`build.gradle`文件中使用此代码如下所示：

```
//自定义插件扩展
class HelloPlugin implements Plugin<Project> {
    void apply(Project project) {
        // Add the 'greeting' extension object
        project.extensions.create("greeting", HelloPluginExtension)

        // Add a task that uses the configuration
        project.task('hello')  {
            doLast{
                println project.greeting.message
            }
        }
    }
}

class HelloPluginExtension {
    String message
}

apply plugin: HelloPlugin
//两种使用方法
/** greeting{
    message="maxuan course"
} **/

greeting.message="maxuan course1"
```

码炫课堂版权所有
大流q群：963060292

使用以下代码执行上述脚本，得到结果如下

```
D:\mypro\gvy>gradle -q hello
maxuan course
```

Gradle为每个扩展对象添加了一个配置闭包，因此可以将分组设置在一起，如下代码， 在`build.gradle`文件中使用此代码。

```
apply plugin: GreetingPlugin

greeting {
    message = 'Hi'
    greeter = 'Gradle'
}

class GreetingPlugin implements Plugin<Project> {
    void apply(Project project) {
```

```
        project.extensions.create("greeting", GreetingPluginExtension)

        project.task('hello') {
          doLast{
            println "${project.greeting.message} from ${project.greeting.greeter}"
          }
        }
    }
}

class GreetingPluginExtension {
    String message
    String greeter
}
```

使用以下代码执行上述脚本，得到结果如下 -

```
D:\mypro\gvy>gradle -q hello
Hi from Gradle
```

码炫课堂版权所有
关注群：963060292

5、标准Gradle插件

在`Gradle`分布中包含不同的插件。

1）语言插件

这些插件的添加，让JVM在编译和执行时对各种语言支持。

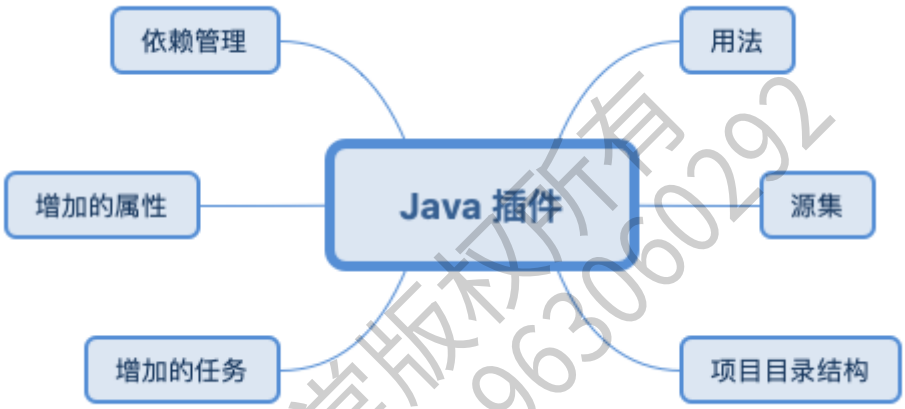
插件Id	自动应用	描述
java	java-base	向项目添加Java编译，测试和绑定的功能。它作为许多其他Gradle插件的基础。
groovy	java,groovy-base	添加对构建Groovy项目的支持。
scala	java,scala-base	添加对构建Scala项目的支持。
antlr	Java	添加了使用Antlr生成解析器的支持。

2）孵化语言插件

这些插件添加对各种语言的支持。

插件Id	自动应用	描述
汇编	—	向项目添加本地汇编语言功能。
c	—	向项目添加C语言的源代码编译功能。
cpp	—	向项目添加C++语言的源代码编译功能。
objective-c	—	向项目添加objective-c语言的源代码编译功能。
objective-cpp	—	向项目添加Objective-C++语言的源代码编译功能。
windows-resources	—	添加本机二进制文件包括Windows资源的支持。

6、Gradle Java插件（重点）



码炫课堂版权所有 963060292 交流Q群

Java 插件是构建 JVM 项目的基础，它为项目增加了很多能力，例如编译，测试，打包，发布等等。

很多插件都是基于 Java 插件实现的，例如 Android 插件。

1）用法

使用 id 应用插件

```
plugins {
    id 'java'
}
```

2）Source sets 源集

Java 插件引入了源集的概念，它在逻辑上表示一组用于编译执行的源文件，这些源文件可能包括源代码文件和资源文件。

一个源集有一个相关联的编译类路径和运行时类路径。

Java 插件就是通过源集的概念来管理源代码目录的。

源集的一个用途是，把源文件进行逻辑上的分组，以描述它们的目的。
例如，你可能会使用一个源集来定义一个集成测试套件，或者你可能会使用单独的源集来定义你的项目的 API 和实现类。

Java 插件提供了两个标准源集：

- main 包含了项目的源代码，被用于编译和生成 JAR 文件
- test 包含单元测试源代码，它们将被编译并使用 JUnit 或 TestNG 来执行

源集提供了很多属性，我这里就列出几个重要的属性：

属性	类型	默认值	描述
name - 只读	String	非空	源集的名字
output - 只读	SourceSetOutput	非空	源集的输出文件，包括它编译过的类和资源。
output.classesDirs 只读	FileCollection	$buildDir/classes/java/$name 例如：build/classes/java/main	源集编译过的 class 文件目录
output.resourcesDir 只读	File	$buildDir/resources/$name 例如main源集：build/resources/main	源集产生的资源目录
java - 只读	SourceDirectorySet	${project.projectDir}/src/${sourceSet.name}/java	源集的 Java 源代码 ，只包含 .java 会排除其他类型。
java.srcDirs	Set	src/$name/java,例如 src/main/java	源集的 Java 源文件的源目录。是一个集合，可以设置多个源代码目录，更改源代码目录就是更改这个属性
java.outputDir	File	$buildDir/classes/java/$name, e.g. build/classes/java/main	源代码编译的 class 文件输出目录
resources - 只读	SourceDirectorySet	${project.projectDir}/src/${sourceSet.name}/resources	源集的资源，只包含资源。
resources.srcDirs	Set	src/$name/resources	源集的资源目录，是一个集合，可以指定多个

跟多的源集属性可以查看下面的文档

sourceSets DSL

码悦课堂版权所有 交流Q群：963060292

sourceSets API

3）定义一个新的源集

源集的位置也很重要，不要在 `dependencies` 下面，否则对源集的依赖就将不起作用

```
sourceSets {
    other
}
```

4）访问源集

sourceSets 是 Java 插件为 Project 增加的一个属性，可以直接使用。

```
task outSourceSet {
    doLast {
        //遍历
        sourceSets.all {
            println "$name -> "
        }
        println "-----split-----"
        //单个的
        println "${sourceSets.main.name} -> "
        println "${sourceSets['main'].name} -> "

        //一些属性
        println " java.srcDirs -->${sourceSets.main.java.srcDirs}"
        println " resource.srcDirs -->${sourceSets.main.resources.srcDirs}"
    }
}
```

5）为源集添加依赖

```
dependencies {
    // This dependency is used by the application.
    implementation 'com.google.guava:guava:27.1-jre'

    // Use JUnit test framework
    testImplementation 'junit:junit:4.12'
    //为 other 源集添加依赖
    otherImplementation 'com.google.code.gson:gson:2.8.5'
}
```

码炫课堂版权所有
交流Q群：963060292

6）将源集打成一个 JAR 包

创建一个 otherJar 任务，将源集的输出作为任务的文件来源。

执行这个任务即可生成 JAR 包。

```
/**
 * 为 other 源集打个 jar 包
 * 默认输出目录是 build/libs
 * 默认名字是 [archiveBaseName]-[archiveAppendix]-[archiveVersion]-
[archiveClassifier].[archiveExtension]
 */
task otherJar(type:Jar){
    archiveBaseName = sourceSets.other.name
    archiveVersion = '0.1.0'
    destinationDirectory = file("${project.projectDir}/jar")
    from sourceSets.other.output
}
```

7）为源集生成 doc

创建一个任务将源集的所有 Java 文件作为源文件。

执行这个任务即可生成 doc 文件。

```
task otherDoc(type:Javadoc){
    destinationDir = file("${project.projectDir}/doc")
    source sourceSets.other.allJava
    title sourceSets.other.name
}
```

码炫课堂版权所有
交流Q群：963060292

8）项目结构

Java 插件的默认目录结构如下所示，无论这些文件夹中有没有内容，Java 插件都会编译里面的内容，并处理没有的内容。

这个目录结构也是 Java 世界标准的项目目录。

目录	描述
src/main/java	Java 源文件目录
src/main/resources	资源文件目录，例如 xml 和 properties 文件
src/test/java	Java 测试源文件目录
src/test/resources	测试资源目录
src/***sourceSet***/java	给定源集的源代码目录
src/***sourceSet***/resources	给定源集的资源目录

9）更改默认目录

这里以更改 main 源集的源代码和资源目录为例

```
sourceSets {
    main {
        java {
            srcDirs = ['src/java']
        }
        resources {
            srcDirs = ['src/resources']
        }
    }
}
```

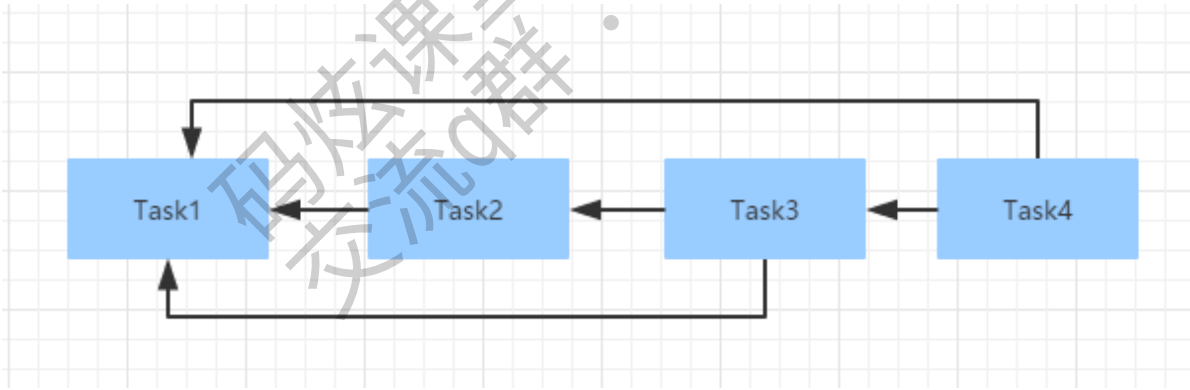
七、Gradle运行构建

Gradle提供了一个命令行来执行构建脚本， 它可以一次执行多个任务，在这里将介绍如何使用不同的选项来执行多个任务。

1、执行多个任务

`Gradle` 可以从单个构建文件执行多个任务。使用 `gradle` 命令处理构建文件。此命令将按列出的顺序编译每个任务，并使用不同的选项执行每个任务以及依赖关系。

示例 - 假设有四个任务 - `task1` ， `task2` ， `task3` 和 `task4` 。 `task3` 和 `task4` 取决于 `task1` 和 `task2` 。 看看下面的图表。



在上面的四个任务是相互依赖的，用一个箭头符号表示。 看看下面的代码。 将其复制并粘贴到 `build.gradle` 文件中。

```
task task1 {
    doLast{
       println 'compiling source #1'
    }
}

task task2(dependsOn: task1) {
    doLast{
       println 'compiling unit tests #2'
    }
}
```

码农课堂版权所有 交流Q群：963060292

```
task task3(dependsOn: [task1, task2]) {
   doLast{
     println 'running unit tests #3'
   }
}

task task4(dependsOn: [task1, task3]) {
    doLast{
      println 'building the distribution #4'
    }
}

1234
```

使用以下代码来编译和执行上述任务，重复依赖的任务只执行一次，如果命令执行成功，将获得以下输出:

```
D:\mypro\gvy>gradle -q task4
compiling source #1
compiling unit tests #2
running unit tests #3
building the distribution #4
```

2、排除任务

要执行中排除某个任务时，可以在 `gradle` 命令中使用 `-x` 选项，并指出要排除的任务的名称。
使用以下命令用于从上面的脚本中排除 `task1` 这个任务。
使用以下代码来编译和执行上述任务。如果命令执行成功，将获得以下输出 -

```
D:\mypro\gvy>gradle task4 -x task1

> Task :task2
compiling unit tests #2

> Task :task3
running unit tests #3

> Task :task4
building the distribution #4

BUILD SUCCESSFUL in 1s
3 actionable tasks: 3 executed
```

3、选择执行哪些构建

当运行gradle命令时，它在当前目录中查找构建文件。我们也可以使用 `-b` 选项选择指定的构建文件的路径。以下示例显示在 `subdir/` 子目录中创建一个新文件 `newbuild.gradle`，并创建一个名称为 `hello` 任务。创建的 `newbuild.gradle` 文件的代码内容如下 -

码悦 保留版权所有 963060292 交流q群

```
task hello {
    doLast{
       println "Use File:$buildFile.name in '$buildFile.parentFile.name'."
    }
}
```

使用以下代码来编译和执行上述任务。如果命令执行成功，将获得以下输出 -

```
D:\mypro\gvy> gradle -q -b subdir/newbuild.gradle hello
Use File: newbuild.gradle in 'subdir'.
```

4、获取构建信息

Gradle提供了几个内置任务来检索有关任务和项目的详细信息，这对理解构建的结构和依赖性以及调试一些问题很有用，可使用项目报告插件向项目中添加任务，来生成这些报告。

1）列出项目

可以使用`gradle -q projects`命令来列出所选项目及其子项目的项目层次结构。下面是一个列出构建文件中的所有项目的示例 -

```
D:\mypro\gvy>gradle -q projects

------------------------------------------------------------
Root project
------------------------------------------------------------

Root project 'gvy'
No sub-projects

To see a list of the tasks of a project, run gradle <project-path>:tasks
For example, try running gradle :tasks
```

码炫课堂版权所有
交流Q群：963060292

报告显示每个项目的描述（如果有指定的话），可以使用以下命令指定描述将其粘贴到`build.gradle`文件中。例如：在build.gradle文件中加上 description 'this is the first project'
再一次执行命令，得到以下结果：

```
D:\mypro\gvy>gradle -q projects

------------------------------------------------------------
Root project - this is the first project
------------------------------------------------------------

Root project 'gvy' - this is the first project
No sub-projects

To see a list of the tasks of a project, run gradle <project-path>:tasks
For example, try running gradle :tasks
```

2) 列出任务

使用以下命令列出属于多个项目的所有任务。如下所示

```
D:\mypro\gvy>gradle -q tasks --all

------------------------------------------------------------
All tasks runnable from root project - The shared API for the application
------------------------------------------------------------

Build Setup tasks
-----------------
init - Initializes a new Gradle build. [incubating]
wrapper - Generates Gradle wrapper files. [incubating]

Help tasks
----------
buildEnvironment - Displays all buildscript dependencies declared in root
project 'script'.
components - Displays the components produced by root project 'script'.
[incubating]
dependencies - Displays all dependencies declared in root project 'script'.
dependencyInsight - Displays the insight into a specific dependency in root
project 'script'.
help - Displays a help message.
model - Displays the configuration model of root project 'script'. [incubating]
projects - Displays the sub-projects of root project 'script'.
properties - Displays the properties of root project 'script'.
tasks - Displays the tasks runnable from root project 'script'.

Other tasks
-----------
task1
task2
task3
task4
```

码炫课堂版权所有
交流q群：963060292

以下是其它一些命令及其说明的列表。

编号	命令	描述
1	gradle –q help –task	提供有关指定任务或多个任务的使用信息（如路径，类型，描述，组）。
2	gradle –q dependencies	提供所选项目的依赖关系的列表。
3	gradle -q api:dependencies — configuration	提供有关配置的有限依赖项的列表。
4	gradle –q buildEnvironment	提供构建脚本依赖项的列表
5	gradle –q dependencyInsight	提供了一个洞察到一个特定的依赖
6	gradle –q properties	提供所选项目的属性列表

八、Gradle构建JAVA项目

本章介绍如何使用Gradle构建文件来构建一个Java项目， 首先，我们必须向构建脚本中添加Java插件，因为它提供了编译Java源代码，运行单元测试，创建Javadoc和创建JAR文件的任务。

在`build.gradle`文件中使用以下代码行：

```
apply plugin: 'java'
```

怎么样，是不是很简单？

1、Java默认的项目布局

每当添加一个插件到构建中，它会假设一个特定的Java项目设置（类似于Maven）。看看下面的目录结构。

- `src/main/java` 目录包含Java源代码；
- `src/test/java` 目录包含测试用的源代码；

如果遵循上面设置，以下构建文件足以编译，测试并捆绑Java项目。
要启动构建，请在命令行上键入以下命令。

```
D:\mypro\gvy> gradle build
```

SourceSets可用于指定不同的项目结构。

例如，指定源代码存储在`src`文件夹中，而不是在`src/main/java`中，例如下面的目录结构：

```
apply plugin: 'java'
sourceSets {
   main {
      java {
         srcDir 'src'
      }
   }
```

码农课堂版权所有 交流Q群：963602

```
    test {
        java {
            srcDir 'test'
        }
    }
}
```

2、初始化任务执行

Gradle还不支持多个项目模板。但它提供了一个init来初始化任务来创建一个新的Gradle项目的结构。如果没有指定其他参数，任务将创建一个Gradle项目，其中包含gradle包装器文件，`build.gradle`和`settings.gradle`文件。

当使用`java-library`作为值并添加`--type`参数时，将创建一个java项目结构，`build.gradle`文件包含带有Junit的某个Java模板。 看看下面`build.gradle`文件的代码。

```
apply plugin: 'java'

repositories {
    jcenter()
}

dependencies {
    compile 'org.slf4j:slf4j-api:1.7.12'
    testCompile 'junit:junit:4.12'
}
```

在仓库(`repositories`)这部分中，它定义了要从哪里找到依赖，Jcenter是为了解决依赖问题， 依赖关系（`dependencies`）部分用于提供有关外部依赖关系的信息。

3、指定Java版本

通常，Java项目要有一个版本和一个目标RE，在其上编译它。`version`和`sourceCompatibility`属性可以在`build.gradle`文件中设置。

```
version = 0.1.1
sourceCompatibility = 1.8
targetCompatibility = 1.8
```

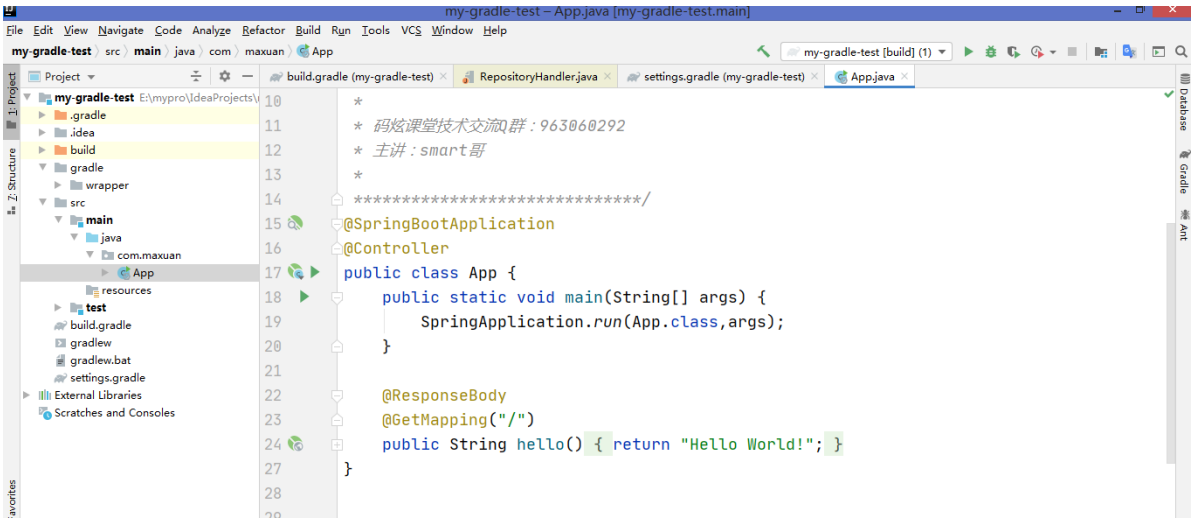
如果这是一个可执行的Java应用程序，`MANIFEST.MF`文件必须要指定具有`main方`法的类。

```
apply plugin: 'java'

jar {
    manifest {
        attributes "Manifest-Version": 1.0,
                'Main-Class': 'com.maxuan.App'
    }
}
```

码悠课堂版权所有 交流Q群：963060292

创建gradle工程，my-gradle-test,在com.maxuan目录下新建App.java,代码如下：




```java
package com.maxuan;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

/*******************************
 *
 * 码炫课堂技术交流Q群：963060292
 * 主讲：smart哥
 *
 *******************************/
@SpringBootApplication
@Controller
public class App {
    public static void main(String[] args) {
        SpringApplication.run(App.class,args);
    }

    @ResponseBody
    @GetMapping("/")
    public String hello() {
        return "Hello World!";
    }
}
```

码炫课堂版权所有 交流Q群：963060292

`build.gradle`文件中部分代码片段如下：

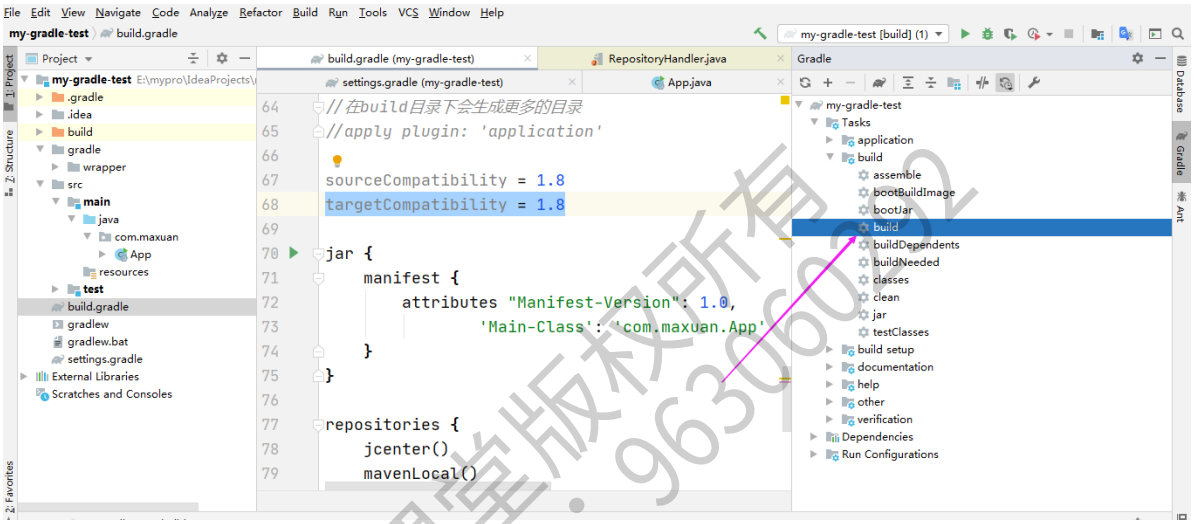
```
apply plugin: 'java'

repositories {
   jcenter()
}
```

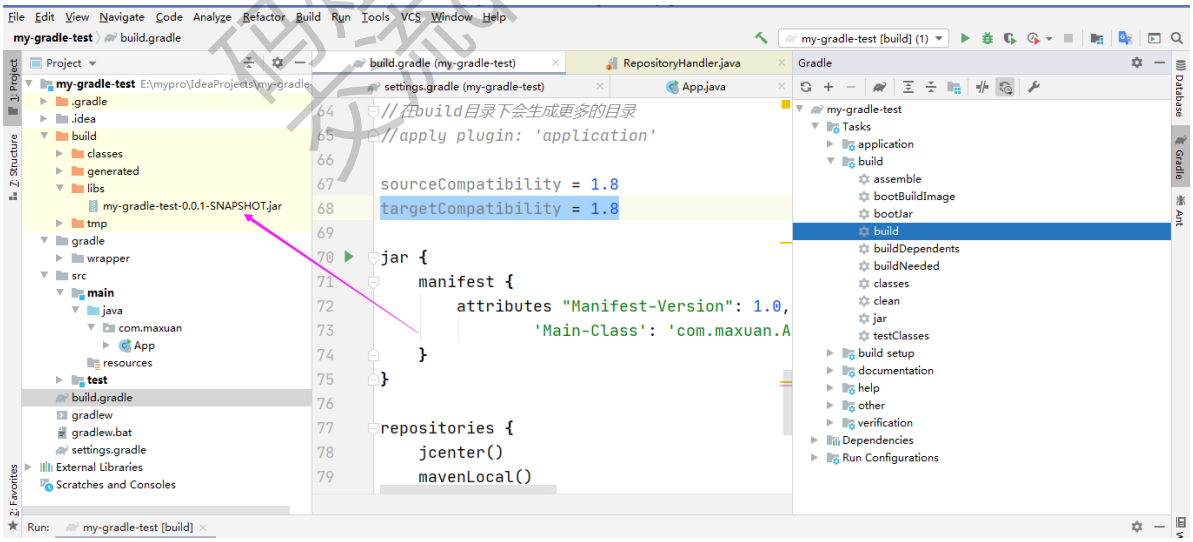
```
dependencies {
    compile 'org.springframework.boot:spring-boot-starter-web'
    testCompile group: 'junit', name: 'junit', version: '4.12'
}

jar {
     manifest {
         attributes "Manifest-Version": 1.0,
                    'Main-Class': 'com.maxuan.App'
     }
}
```

执行 gradle build：



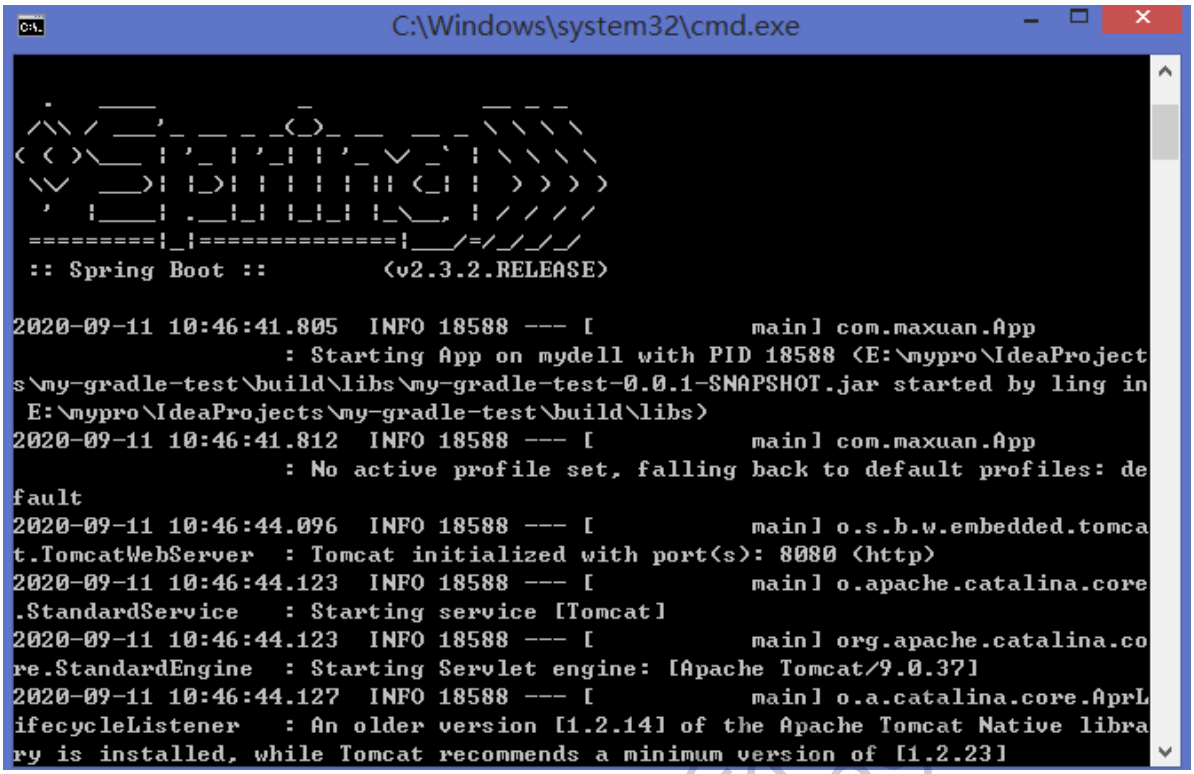
执行完毕之后在 build/libs下生成一个jar文件，该文件是可执行jar，如下图所示：



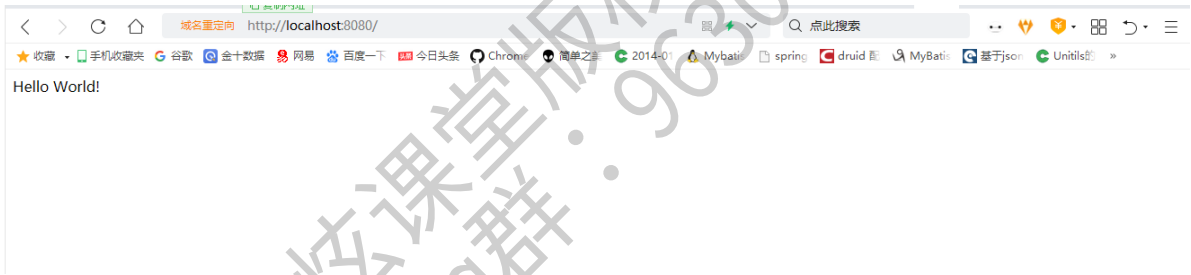
cmd到libs下，执行jar包，

```
E:\mypro\IdeaProjects\my-gradle-test\build\libs>java -jar my-gradle-test-0.0.1-S
NAPSHOT.jar
```

结果如下：



springboot工程成功启动，打开浏览器输入：http://localhost:8080/ 回车，打印“Hello World”



码炫课堂版权所有
交流Q群：963060

九、Gradle构建Groovy项目

本章介绍如何使用`build.gradle`文件编译和执行Groovy项目。

1、Groovy插件

Gradle的Groovy插件扩展了Java插件，并为Groovy程序提供了任务。可以使用以下行来应用groovy插件。

```
apply plugin: 'groovy'
```

完整的构建脚本文件如下。将以下代码复制到`build.gradle`文件中。

```
apply plugin: 'groovy'

repositories {
   mavenCentral()
}

dependencies {
   compile 'org.codehaus.groovy:groovy-all:2.4.5'
   testCompile 'junit:junit:4.12'
}
```

可以使用以下命令来执行构建脚本。

```
gradle build
```

2、Groovy项目的默认项目布局

Groovy插件假定Groovy项目有手动做过一定的设置。

- `src/main/groovy` 包含Groovy源代码；
- `src/test/groovy` 包含Groovy测试源代码；
- `src/main/java` 包含Java源代码；
- `src/test/java` 包含Java测试源代码；

在 `build.gradle` 文件所在当前目录下的 `build` 文件夹中查看相应的目录，是否执行后有生成文件。

十、Gradle测试

测试任务会自动检测和执行测试源集合中的所有单元测试，它还会在测试执行完成后生成报告，JUnit和TestNG都是支持的API。

1、测试检测

测试任务通过检查编译的测试类来检测哪些类是测试类， 默认情况下，它扫描所有 `.class` 文件，不过也可以设置自定义包含/排除，只有那些类才会被扫描。

根据所使用的测试框架（ `JUnit` / `TestNG` ），测试类检测使用不同的标准。

如果不想使用测试类检测，可以通过将 `scanForTestClasses` 设置为 `false` 来禁用它。

```
test{
  //默认使用junit
  useJUnit{}
  //使用testNG
  useTestNG()
}
```

码炫课堂版权所有
交流q群：963060292

2、包括和排除指定测试

`Test`类有一个`include`和`exclude`方法。这些方法可以用于指定哪些测试应该运行。

禁用测试

```
test {
    enabled = false
}
```

只运行包含的测试

```
test {
   include 'com/maxuan*/*'
}
```

跳过排除的测试

```
test {
   exclude 'com/maxuan*/*'
}
```

以下代码中所示的`build.gradle`示例文件显示了不同的配置选项。

```
apply plugin: 'java'
// adds 'test' task
test {
   // enable TestNG support (default is JUnit)
   useTestNG()
   //scanForTestClasses = false
   // set a system property for the test JVM(s)
   systemProperty 'some.prop', 'value'

   // explicitly include or exclude tests
   include 'com/maxuan/**'
   exclude 'com/maxuan1/**'

   // show standard out and standard error of the test JVM(s) on the console
   testLogging.showStandardStreams = true

   // set heap size for the test JVM(s)
   minHeapSize = "64m"
   maxHeapSize = "512m"

   // set JVM arguments for the test JVM(s)
   jvmArgs '-XX:MaxPermSize=256m'

   // listen to events in the test execution lifecycle
   beforeTest {
      descriptor -> logger.lifecycle("Running test: " + descriptor)
   }

   // listen to standard out and standard error of the test JVM(s)
```

码农课堂版权所有 交流Q群：963060292

```
    onOutput {
        descriptor, event -> logger.lifecycle
            ("Test: " + descriptor + " produced standard out/err: "
            + event.message )
    }
}
```

可以使用以下命令语法来执行一些测试任务。

```
gradle <someTestTask> --debug-jvm
```

十一、Gradle多项目构建

`Gradle`可以轻松处理各种大小规模的项目，小项目由一个单一的构建文件和一个源代码树构成，大项目可以将其拆分成更小的，相互依赖的模块，以便更容易理解，Gradle完美支持这种多项目构建的场景。

1、多项目构建的结构

这种构建有各种形状和大小，但它们都有一些共同的特点 -

- 在项目的根目录或主目录中都有一个`settings.gradle`文件。
- 根目录或主目录都有一个`build.gradle`文件。
- 具有自己的`*.gradle`构建文件的子目录（某些多项目构建可能会省略子项目构建脚本）。

要列出构建文件中的所有项目，可以使用以下命令。

```
E:\mypro\IdeaProjects\my-gradle-all>gradle -q projects
```

如果命令执行成功，将获得以下输出。

```
E:\mypro\IdeaProjects\my-gradle-all>gradle -q projects

------------------------------------------------------------
Root project
------------------------------------------------------------

Root project 'my-gradle-all'
+--- Project ':my-gradle-entry'
\--- Project ':my-gradle-service'

To see a list of the tasks of a project, run gradle <project-path>:tasks
For example, try running gradle :my-gradle-entry:tasks
```

报告将显示每个项目的描述（如果指定），可以使用以下命令指定描述,将其粘贴到`build.gradle`文件中。

```
description = 'The shared API for the application'
```

码农课堂版权所有 交流Q群：963060292

2、指定常规构建配置

在根项目中的`build.gradle`文件中，常规配置可以应用于所有项目或仅应用于子项目。

```
allprojects {
    group = 'com.maxuan'
    version = '0.1.0'
}

subprojects {
    apply plugin: 'java'
}
```

这指定了一个公共`com.maxuan`组和一个`0.1.0`版本到所有项目，`subprojects`闭合所有应用对子项目通用配置，但不对根项目应用，如：`allprojects`闭合。

3、项目指定配置和依赖关系

核心`my-gradle-entry`和`my-gradle-service`子项目也可以有自己的`build.gradle`文件，如果它们有特定的需求，那么一般不会应用根项目配置。

例如，my-gradle-service项目通常具有对核心项目的依赖性,所以在my-gradle-m1项目中需要有配置自己的`build.gradle`文件来指定这个依赖。

```
dependencies {
    compile project(':core')
    compile 'log4j:log4j:1.2.17'
}
```

项目依赖项可使用项目方法指定。

4、Gradle多项目构建的示例

1）定义公共行为

让我们看看下面的一个例子的项目树。这是一个多项目构建，其中包含一个名为`my-gradle-all`的根项目和一个名称为`my-gradle-entry`和`my-gradle-service`的子项目。

多项目树 - `my-gradle-all`和`my-gradle-entry`和`my-gradle-service`项目的构建布局如下图所示：

```
Root project 'my-gradle-all'
    build.gradle
    settings.gradle
+--- Project ':my-gradle-entry'
\--- Project ':my-gradle-service'
```

首先，创建一个文件 settings.gradle 并写入以下代码内容（自动写入）

```
include 'my-gradle-service'
include 'my-gradle-entry'
```

在根项目的`build.gradle`并写入以下代码：

码炫课堂版权所有
交流Q群：963060232

```
allprojects {
    task hello {
        doLast{
            task -> println "I'm $task.project.name"
        }
   }
}
```

并执行 `gradle -q hello` 输出结果如下 -

```
E:\mypro\IdeaProjects\my-gradle-all>gradle -q hello
I'm my-gradle-all
I'm my-gradle-entry
I'm my-gradle-service
```

这是如何工作的？ Project API提供了一个属性`allprojects`，它返回当前项目及其下面所有子项目的列表。 如果使用闭包调用`allprojects`，则闭包的语句将委派给与所有项目相关联的项目，当然也可以通过`allprojects.each`进行迭代，但这将更冗长。

其他构建系统使用继承作为定义公共行为的主要方法，Gradle也为项目提供继承，但Gradle使用配置注入作为定义公共行为的常用方式，这是一种非常强大和灵活的配置多项目构建的方式，共享配置的另一种可能性是使用公共外部脚本。

5、子项目配置

`Project API`还提供了一个仅用于访问子项目的属性。

1) 定义公共行为

定义所有项目和子项目的公共行为，编辑`build.gradle`文件使用以下代码 -

```
allprojects {
    task hello {
        doLast{
           task -> println "I'm $task.project.name"
        }
    }
}

subprojects {
    hello {
        doLast{
           println "- I depend on my-gradle-all"
        }
    }
}
```

并执行 `gradle -q hello` 输出结果如下 -

码炫课堂版权所有 交流q群：963060292

```
D:\mypro\IdeaProjects\my-gradle-all>gradle -q hello
I'm my-gradle-all
I'm my-gradle-entry
- I depend on my-gradle-all
I'm my-gradle-service
- I depend on my-gradle-all
```

注意两个代码片段引用“`hello`”任务, 第一个，它使用“`task`”关键字，构建任务并提供它的基本配置。

第二部分不使用“`task`”关键字，因为它进一步配置现有的“`hello`”任务,只能在项目中构建一次任务，但可以添加任意数量的代码块以提供其他配置。

2）添加指定行为

可以在常见行为之上添加指定的行为，要应用这个特定的行为，通常将项目特定的行为放在项目的构建脚本中，我们可以为 `bluewhale` 项目添加项目特定的行为，如下所示：
编辑 `build.gradle` 文件使用以下代码：

```
allprojects {
    task hello {
        doLast{
           task -> println "I'm $task.project.name"
        }
    }
}

subprojects {
    hello {
        doLast{
           println "- I depend on my-gradle-all"
        }
    }
}

project(':my-gradle-service').hello {
    doLast{
        println "- I'm my-gradle-service and depends on my-gradle-all."
    }
}
```

码炫课堂版权所有
交流q群：963060292

并执行 `gradle -q hello` 输出结果如下：

```
E:\mypro\IdeaProjects\my-gradle-all>gradle -q hello
I'm my-gradle-all
I'm my-gradle-entry
- I depend on my-gradle-all
I'm my-gradle-service
- I depend on my-gradle-all
- I'm my-gradle-service and depends on my-gradle-all.
```

正如上面所说的，通常把项目特定的行为放入这个项目的构建脚本中。

3）定义 `my-gradle-service` 项目的具体行为

构建布局如下图中所示:

```
my-gradle-all/
  build.gradle
  settings.gradle
  my-gradle-service/
    build.gradle
  my-gradle-entry/
    build.gradle
```

`settings.gradle` 文件的内容：

```
rootProject.name = 'my-gradle-all'
include 'my-gradle-service'
include 'my-gradle-entry'
```

`my-gradle-service/build.gradle` 文件的内容 ：

```
hello.doLast {
  println "- I'm the my-gradle-m1 module."
}
```

`my-gradle-entry/build.gradle` 文件的内容：

```
hello.doLast {
  println "- I'm the my-gradle-entry module."
}
```

`build.gradle` 文件的内容：

```
allprojects {
    task hello {
        doLast{
            task -> println "I'm $task.project.name"
        }
    }
}

subprojects {
    hello {
        doLast{
            println "- I depend on my-gradle-all"
        }
    }
}
```

并执行 `gradle -q hello` 输出结果如下 -

码农课堂版权所有
交流q群：963060292

```
E:\mypro\IdeaProjects\my-gradle-all>gradle -q hello
I'm my-gradle-all
I'm my-gradle-entry
- I depend on my-gradle-all
I'm my-gradle-service
- I depend on my-gradle-all
- I'm the my-gradle-service module.
```

十二、Gradle部署

Gradle提供了几种部署构建工件(artifacts)存储库的方法。将工件的签名部署到Maven仓库时，还需要签署已发布的 `POM` 文件。

1、使用Maven插件发布

Gradle默认提供 `maven-publish` 插件， 它用于发布 `gradle` 脚本， 看看下面的代码。

```
apply plugin: 'java'
apply plugin: 'maven-publish'
publishing {
    publications {
        maven(MavenPublication) {
            from(components.java)
        }
    }

    repositories {
        maven {
            url "$buildDir/repo"
        }
    }
}
```

码炫课堂版权所有
交流q群：963060292

当应用[Java]和 `maven-publish` 插件时，有几个发布选项， 看看下面的代码，它会将项目部署到远程仓库。

```
apply plugin: 'groovy'
apply plugin: 'maven-publish'
group 'com.maxuan'
version = '1.0.0'
publishing {
   publications {
      mavenJava(MavenPublication) {
         from components.java
      }
   }

   repositories {
      maven {
         default credentials for a nexus repository manager
         credentials {
```

```
            username 'admin'
            password 'mypasswd'
         }          // 发布maven存储库的url
         url "http://localhost:8080/nexus/content/repositories/releases/"
      }
   }
}
```

2、将项目从Maven转换为Gradle

有一个特殊的命令用于将Apache Maven `pom.xml`文件转换为`Gradle`构建文件，如果此任务已经知道使用的所有`Maven`插件。

在本节中，以下`pom.xml`的`maven`配置将转换为`Gradle`项目。创建一个`D:/pom.xml`并使用下面的代码。

```
<project xmlns = "http://maven.apache.org/POM/4.0.0"
   xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation = "http://maven.apache.org/POM/4.0.0
   http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>com.example.app</groupId>
   <artifactId>example-app</artifactId>
   <packaging>jar</packaging>

   <version>1.0.0-SNAPSHOT</version>

   <dependencies>
      <dependency>
         <groupId>junit</groupId>
         <artifactId>junit</artifactId>

         <version>4.11</version>
         <scope>test</scope>
      </dependency>
   </dependencies>
</project>
```

码悦课堂版权所有
交流Q群：963060292

可以命令行上使用以下命令，然后生成以下`Gradle`配置内容。

```
D:/> gradle init --type pom
Starting a Gradle Daemon, 1 incompatible and 1 stopped Daemons could not be
reused, use --status for details
:wrapper
:init
Maven to Gradle conversion is an incubating feature.

BUILD SUCCESSFUL

Total time: 11.542 secs
```

`init`任务依赖于包装器任务，因此它创建了一个`Gradle`包装器。
生成的`build.gradle`文件类似于以下内容。

```
apply plugin: 'java'
apply plugin: 'maven'

group = 'com.maxuan'
version = '1.0.0-SNAPSHOT'

description = "this is reverse gradle project"

sourceCompatibility = 1.8
targetCompatibility = 1.8

repositories {

     maven { url "http://repo.maven.apache.org/maven2" }
}
dependencies {
    testCompile group: 'junit', name: 'junit', version:'4.11'
}
```

十三、IDEA中创建Gradle工程

见视频讲解

码炫课堂技术交流群：963060292

群名称:码炫课堂java架构群1
群　号:963060292

码炫课堂版权所有 交流群963060292