# 第一部分 Gradle介绍

Gradle是Java软件开发中的自动化构建工具，类似于传统工具如Ant和Maven。Gradle吸收或沿用了Maven中比较成功的一些实践，但相对于Maven或Ant又有极强的扩展。

Gradle从已有构建工具中吸取经验，取其精华，并进一步提升。

传统的面向Java项目的基建工具有两个：Ant和Maven，Ant是最早的工具

Ant和Maven构建逻辑都是使用XML描述，XML是非常好的层级数据描述语言，但是对于描述程序流程和构建逻辑却存在不足之处，随着构建脚本复杂的增加，维护构建代码就成为了噩梦。

## 1、项目自动化介绍

### 1.1 没有项目自动化的生活

### 1.2 项目自动化的好处

#### 1.2.1 防止手动介入

手动地执行每一步去实现和交付软件是耗时且易于犯错的。

#### 1.2.2 创建可重复的构建

软件的构建通常都有预定义和有序的步骤，比如，你需要先编译源代码，然后运行测试，最后组装可交付软件。

#### 1.2.3 让构建便携

你已经看到，能够在IDE中运行的构建是非常有限的。首先，你必须将特定的产品安装在机器上，其实，IDE也许只适用于某一操作系统。一个自动化构建不应该依赖于特定的运行环境才能工作，无论是操作系统还是IDE。最佳的方式应该是，自动化任务从命令行运行，它允许你在任何时间和任何一台想要运行构建的机器上运行。

### 1.3 项目自动化的类型

在本章的开始，你看到了用户可以请求一个构建运行。用户可以是任何想要触发该构建的利益相关者，比如开发人员、测试人员或者产品拥有者。

#### 1.3.1 按需构建

按需构建的典型用例就是在用户自己的机器上触发构建。

在大多数情况 下，用户在命令行执行一个脚本去运行预先定义的有序任务，比如，编译源代码，拷贝文件从A目录到B目录，或者组装一个可交付软件，通常，这种类型的构建一天要运行多次。

#### 1.3.2 触发构建

例如，每天凌晨1：00点或者每隔15分钟。和所有的定时作业一样，预定的自动化任何通常运行在一个专用的服务器上。这种类型的构建特别适用于生成报告或者项目的文件操作。

实现预定义和触发构建的实践方式通常也叫作持续集成（CI）。

#### 1.3.3 预定构建

### 1.4 构建工具

#### 1.4.1 什么是构建工具

#### 1.4.2 构建工具剖析

通常情况下，会使用脚本语言来表达构建逻辑，这就是个构建也叫作构建脚本的原因。

### 1.5 Java 构建工具

#### 1.5.1 Apache Ant

Apache Ant（AnotherNeat Tool）是一个用Java编写的开源构建工具，其主要目的是在Java项目中为常用任务提供自动化，例如编译源代码、运行单元测试、打包jar文件和生成Javadoc文档。

虽然Ant的核心是用Java编写的，但是build文件是通过XML表示的，这样就可以在任何运行时环境下使用了。Ant不提供依赖管理器，所以你需要自己管理外部依赖。然而，Ant可以和另一个Apache的项目Ivy很好地集成，它是一个完善且独立的依赖管理器。

一个Ant构建脚本由三个基本元素组成：一个project、多个target和可用的task。Task是一段可执行的代码，例如创建一个目录或者移动一个文件。

target是你想要执行的task的一个集合。

project是Ant脚本的顶级元素，包含一个或多个target，在每个构建脚本中，必须且只能定义一个project

Ant缺点：相比其他更简明的定义语言，会导致构建脚本过于臃肿和啰唆。 复杂的构建逻辑会长又难以维护的构建脚本，当尝试用xml去描述类似if-then-else的逻辑语句时，它完全就成了一种负担。在没有Ivy的情况下，使用Ant很难管理依赖。

#### 1.5.2 Apache Maven

Maven为你的项目配置和行为提供了有意义的默认值。

Maven缺点：为Maven写定制的扩展过于累赘

#### 1.5.3 对下一代构建工具的需求

### 1.6 总结

对于开发和测试人员来说，没有项目自动化的工作，都是重复、单调和易犯错的。伴随着软件交付过程中的每一步——从源码编译到打包软件，再到发布至测试和产品环境都必须手动操作。项目自动化帮你消除手动操作介入的负担，让你的团队更有效率，带你进入 一个一键式和故障安全的软件发布过程。

## 2、下一代构建工具：Gradle

多年来，构建只有编译和打包的简单需求。但是现代软件开发的规模改变了，因此有了对自动化构建的需求。

Gradle是基于JVM构建工具的新一代版本。

### 2.1 为什么要用Gradle，为什么是现在

#### 2.1.1 Java构建工具的演变

Ant的第一个正式版本是在2000年发布的。

Maven 1发布于2004年7月。

Maven 2发布于2005年10月。

Gradle结合了其它构建工具的最佳特性：

Ant：灵活性、完全控制、目标链

Ivy：依赖管理

Maven：约定优于配置、多模块项目、插件扩展

Gant：在Ant基础上提供Groovy的DSL

#### 2.1.2 为什么选择Gradle

不要改变 一个正在运行的系统，你说呢？你的团队已经花费大量时间来建立项目构建代码基础设施。Gradle式不强迫你完全迁移所有的构建逻辑。它和其他构建工具如Ant和Maven有非常好的集成，这是Gradle优先级列表中的最高优先级。

Gradle很早就有采纳者了，甚至在1.0版本发布之前。像Groovy和Hibernate这样流利的开源项目已经完全切换到Gradle并作为它们构建的支柱。

### 2.2 Gradle引人注目的特性集

Gradle的优点：

* Gradle就是Groovy
* 具有表达性的语言和强大的API
* 和其他构建工具的集成
* 社区驱动和公司支持
* 毫不费力的扩展
* 灵活的约定
* 鲁棒和强大的依赖管理
* 可扩展的构建

#### 2.2.1 可表达性的构建语言和底层的API

一个构建脚本直接映射到Gradle API中Project类的一个实例。Gradle的API文档：<https://docs.gradle.org/current/javadoc/index.html> 。谁会知道？实际上你是在和代码打交道，在完全不知情的情况下，在内存中生成了一个代表构建逻辑的对象。在Gradle脚本中每个元素都有一个与Java类一对一的映射。

Gradle构建语言指南：<http://www.gradle.org/docs/current/dsl/index.html>

#### 2.2.2 Gradle就是Groovy

XML很难定义复杂的逻辑，Gradle的DSL是由Groovy实现的，它提供了基于Java的语法糖，结果就是产生了一种具有可读性和表达性的构建语言。所有你写的脚本都是Groovy。

#### 2.2.3 灵活的约定

对于一个java项目，Gradle已经提供了默认的有意义的任务。Maven非常武断，它建议一个工程只包含一个Java源代码目录且只产生一个jar文件。而Ant不会给你任何关于如何组织构建脚本方面的指导以确保最大的灵活性。Gradle进行了折中，Gradle即提供约定又给予你改变约定的能力

#### 2.2.4 鲁棒和强大的依赖管理

#### 2.2.5 可扩展的构建

#### 2.2.6 轻松的可扩展性

#### 2.2.7 和其它构建工具集成

#### 2.2.8 社区和公司驱动

Gradle基于Apache License 2.0，是免费使用的。在2008年4月第一次发布之后，一个充满生气的社区围绕它快速形成。

Gradle社区论坛是：<http://forums.gradle.org/gradle> 或<https://discuss.gradle.org/>

Gradleware是Gradle背后的技术服务和支持公司。

#### 2.2.9 锦上添花：额外的特性

难道你不讨厌给不同的项目安装新的运行时环境？Gradle包装器是救星！它允许你在任何需要运行构建的机器上从一个指定的仓库下载和安装一个Gradle运行时的新拷贝。

### 2.3 更大的场景：持续交付

能够构建源代码仅仅是软件交付过程中的一个方面。更重要的是，你想要将产品发布到产品环境以创造业务价值。在这个过程 中，你想要运行测试，构建发布产品，分析代码以保证质量，也可能要指定一个目标环境并部署。

自动化的整个过程好处太多了，最重要的是，手动交付软件是很漫长的、易出错而且让人神经紧张。我确信我们中的每个人都会因为一个部署出错要整夜待命。

随着敏捷开发方法的出现，开发团队能够更快地交付软件。2 ~ 3周的发布周期变得很平常。

#### 2.3.1 从构建到部署自动化项目

### 2.4 安装Gradle

下载地址：<https://gradle.org/install/>,包含旧版本下载：https://gradle.org/releases或者<http://services.gradle.org/distributions/>

手动安装：下载如“gradle-4.9-all.zip”，解压，然后设置GRADLE\_HOME，在Path变量中设置%GRADLE\_HOME%\bin，然后在命令行中检测安装是否成功：gradle – v

### 2.5 开始使用Gradle

每个Gradle构建都是以一个脚本开始的。Gradle构建脚本默认的名字是build.gradle。当在shell命令中执行gradle命令时，Gradle会去寻找名字是build.gradle的文件，如果找不到，就会显示一个帮助信息。

让我们在Gradle中实现经典的“Hello World！”的例子。build.gradle文件如下：

task helloWorld {

doLast {

println 'Hello world!'

}

}

在命令行中运行：gradle -q helloWorld

通过 -q定义可选命令行选项quiet，告诉Gradle只输出该task相关的信息。

在完全不需要了解Gradle的情况下，你已经开始使用Gradle的DSL了。Task和action是这门语言的重要元素。名字为doLast的action几乎自表达了它的含义。它是task执行的最后一个action。Gradle还允许使用一种更精简的方式来指定相同的逻辑，使用左移<<来简单的代表doLast。如下：

task helloWorld << {

println 'Hello world!'

}

下面的Demo展示了一些更高级的特性，通过一点点练习，让我们增强对Gradle的信心：

动态任务定义和任务链：

### 2.6使用Gradle的命令行

task startSession << {

chant()

}

def chant() {

ant.echo(message: 'Repeat after me...')

}

3.times {

task "yayGradle$it" << {

println 'Gradle rocks'

}

}

yayGradle0.dependsOn startSession

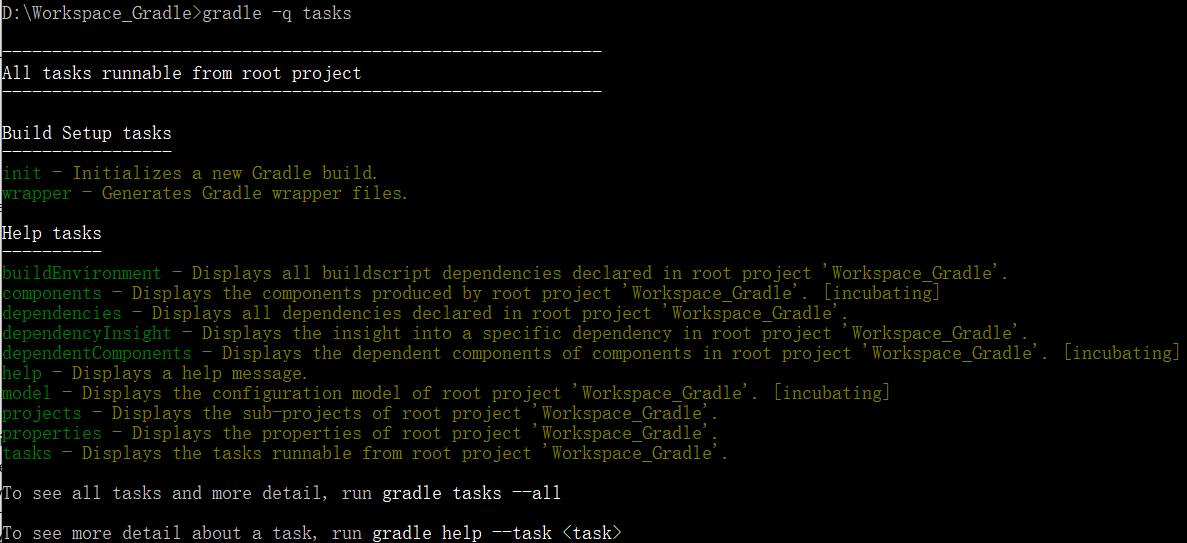
yayGradle2.dependsOn yayGradle1, yayGradle0

task groupTherapy(dependsOn: yayGradle2)

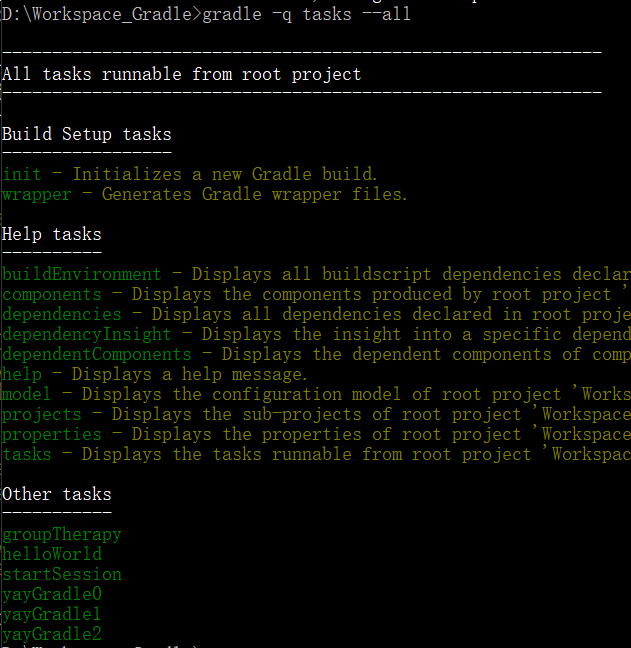
#### 2.6.1 例出项目中所有可用的task

执行命令：gradle -q tasks

如下：



要想查看我们写的task，加上—all选项，如下：



关于输出，有几点需要说明。Gradle提供了任务组的概念，你可以把它看作是多个task的集群。每个构建脚本都会默认暴露Help tasks任务组。如果某个task不属于一个任务组，那么它就会显示在Other tasks中。

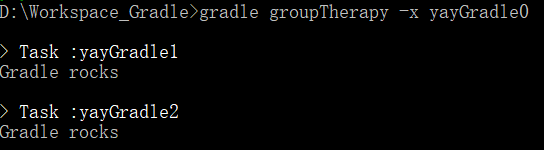
#### 2.6.2 任务执行

执行：gradle yayGradle0 groupTherapy ，虽然这里指定了执行两个任务，但是每个任务只会输出一次，跟之前的效果一样，正确的顺序被保存并且每个任务只执行一次。

可以以缩写的方法执行任务，但是任务的这个缩写必须是唯一的，如：gradle gt等同于gradle groupTherapy

**在执行时排除一个任务**

有时候你想要在构建运行时排除一个指定任务，Gradle提供了一个命令行选项 -x来实现。假设你想要排除任务yayGradle0：gradle groupTherapy -x yayGradle0，输出结果如下：



Gradle排除了任务yayGradle0和它的依赖任务startSession，这个概念叫作智能排除。

#### 2.6.3 命令行选项

Gradle允许你同时定义一个或者多个选项。假设你想要将日记级别改变到INFO，则可以使用-i选项，或者如果想打印出在执行期间发生错误的堆栈跟踪信息，则可以使用-s选项，如：gradle groupTherapy -is或gradle groupTherapy -i -s。

-h：(help) 查看到所有可用的选项

-b：(build file) Gradle构建脚本默认命名约定为build.gradle，使用这个命令选项可以执行一个其它名字的构建脚本，如：gradle -b test.gradle

--offline:通常，构建中声明的依赖必须在离线仓库中存在才可用。如果这些依赖在本地缓存中没有，那么 运行在一个没有网络连接环境中的构建都会失败。使用这个选项可以让你以离线模式运行构建，仅仅在本地缓存 中检查依赖是否存在。

**参数选项**

-D：(system property) Gradle是以一个JVM进程运行的。和所有的Java进程一样，你可以提供一个系统 参数，就像-Dmyprop=myvalue这样。

-p：(project property) 项目参数是构建脚本中可用的变量。你可以使用这个选项直接向构建脚本中传入参数，如-Pmyprop=myvalue

**日志参数**

-i：(info) 在默认设置中，Gradle构建不会提供大量的输出信息。通过 这个选项可以将Gradle的日志级别改变到INFO以获取更多信息。如果你想知道构建中发生了什么，这个选项非常有用。

-s：(stacktrace) 如果构建在运行中出现错误，你会想要知道错误是从哪里开始的。-s选项在有异常抛出时会打印出简短的堆栈跟踪信息，帮助你进行调试。

-q：(quiet) 减少构建出错时打印出来的错误日志信息。

**帮助任务**

tasks: 显示项目中运行的task，包括它们的描述信息。项目中应用的插件可能会提供一些额外的task。

Properties: 显示出项目中所有可用的属性。某些属性是由Gradle的project对象提供的，project对象是一个构建的本质表现形式。其他的属性都是用户定义的，要么来自于属性文件或者命令行选项，要么是直接在构建脚本中定义的。

#### 2.6.4 Gradle守护进程

每次初始化一个构建时，JVM都要启动一次，Gradle的依赖要载入到类加载器中，还要建立项目对象模型。这个过程需要花上好几秒钟。Gradle守护进程是这个问题的救星！

守护进程以后台进程方式运行Gradle。一旦启动，Gradle命令就会在后续的构建中重用之前创建的守护进程，以避免启动时造成的开销。让我们回到之前的构建脚本例子中，在我的机器 上，成功地完成运行groupTherapy任务要花上3秒钟，我们希望提高启动和执行的效率，在命令行中启动Gradle守护进程很简单：在运行gradle命令时加上—daemon选项（如：gradle groupTherapy –daemon）。你可能会注意到在启动守护进程时也花费一点点额外的时间。为了验证守护进程在运行，你可以查看系统进程列表.

守护进程只会被创建一次，即便你在命令行中加了—daemon选项。守护进程会在3小闲时间之后自动过期。任何时候你都可以选择在执行构建时不使用守护进程，只需要添加命令行选项—no-daemonvcb sk ，要手动停止守护进程，可以执行gradle –stop命令。

## 3、通过范例学习构建Gradle项目

当开始开发一个全新的应用时，Java并没有指导你如何项目结构。你可能会问自己在哪里放源文件、配置文件和类库文件。如果想要将应用程序代码与测试代码分开，应该怎么办？Gradle通过引入预定义的项目布局为类似Java的项目提供了基于约定的构建方法。被那种有不同目录结构的遗留应用困住了？没问题！Gradle可以根据你的需要调整约定。

### 3.1 介绍学习案例

一个To Do应用程序

#### 3.1.1 To Do应用程序

#### 3.1.2 任务管理用例

#### 3.1.3 检查组件交互

#### 3.1.4 构建应用功能

**To Do模型类**

每一个ToDoItem类的实例都to do列表中的一个执行项目。属性id是每一个项目的唯一标识，也让你能够在内存中读取和存储它。另外，领域模型类暴露了name和completed属性。为了使代码简洁，getter和setter方法从下面的代码片段中省去了：

public class ToDoItem implements Comparable<ToDoItem> {

private Long id;

private String name;

private boolean completed;

public int compareTo(ToDoItem other) {

return id.compareTo(other.id)

}

}

**领域模型在内存中的持久化**

仓库类接口

package com.manning.gia.todo;

import com.manning.gia.todo.model.ToDoItem;

import java.util.Collection;

public interface ToDoRepository {

List<ToDoItem> findAll();

ToDoItem findById(Long id);

Long insert(ToDoItem toDoItem);

void update(ToDoItem toDoItem);

void delete(ToDoItem toDoItem);

}

To do项目的持久化类

package com.manning.gia.todo.repository;

import com.manning.gia.todo.model.ToDoItem;

import java.util.Collections;

import java.util.ArrayList;

import java.util.List;

import java.util.concurrent.ConcurrentHashMap;

import java.util.concurrent.ConcurrentMap;

import java.util.concurrent.atomic.AtomicLong;

public class InMemoryToDoRepository implements ToDoRepository {

private AtomicLong currentId = new AtomicLong();

private ConcurrentMap<Long, ToDoItem> toDos = new ConcurrentHashMap<>()

@Override

public List<ToDoItem> findAll() {

List<ToDoItem> toDoItems = new ArrayList<ToDoItem>(toDos.values());

Collections.sort(toDoItems);

return toDoItems;

}

@Override

public ToDoItem findById(Long id) {

return toDos.get(id);

}

@Override

public Long insert(ToDoItem toDoItem) {

Long id = currentId.incrementAndGet();

toDoItem.setId(id);

toDos.putIfAbsent(id, toDoItem);

return id;

}

@Override

public void update(ToDoItem toDoItem) {

toDos.replace(toDoItem.id, toDoItem);

}

@Override

public void delete(ToDoItem toDoItem) {

toDos.remove(toDoItem.id);

}

}

**应用程序入口**

实现主体类：

package com.manning.gia.todo;

import com.manning.gia.todo.utils.CommandLineInput;

import com.manning.gia.todo.utils.CommandInputHandler;

public class ToDoApp {

public static final char DEFAULT\_INOUT = '\u0000';

public static void main(String[] args) {

CommandLineInputHandler commandLineInputHandler = new CommandLineInputHandler();

char command = DEFAULT\_INOUT;

while (CommandLineInput.EXIT.getShortCmd() != command) {

commandLineInputHandler.printOptions();

String input = commandLineInputHandler.readInput();

char[] inputChars = input.length() == 1 ? input.toCharArray() : new char[] { DEFAULT\_INOUT };

command = inputChars[0];

CommandLineInput commandLineInput = CommandLineInput.getCommandLineInputForInput(command);

commandLineInputHandler.processInput(commandLineInput);

}

}

}

### 3.2 构建Java项目

要组装一个可执行程序，需要编译源代码，class文件需要打包成Jar文件，JDK提供了像javac和jar这样的工具来帮助实现这些任务。除非你是受虐狂，否则你绝对不会想要一次次手动执行这些任务。

Gradle插件作为驱动力能够自动化这些任务。插件通过引入特定领域的约定和任务来扩展你的项目。Java插件是Gradle自身装载的一个插件。Java插件提供的基本功能远比源代码编译和打包多。它为你的项目建立了一个标准的项目布局，并确保有意义、有顺序地执行任务。

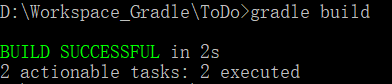
#### 3.2.1 使用Java插件

创建build.gradle文件，并像下面这样告诉它使用Java插件：

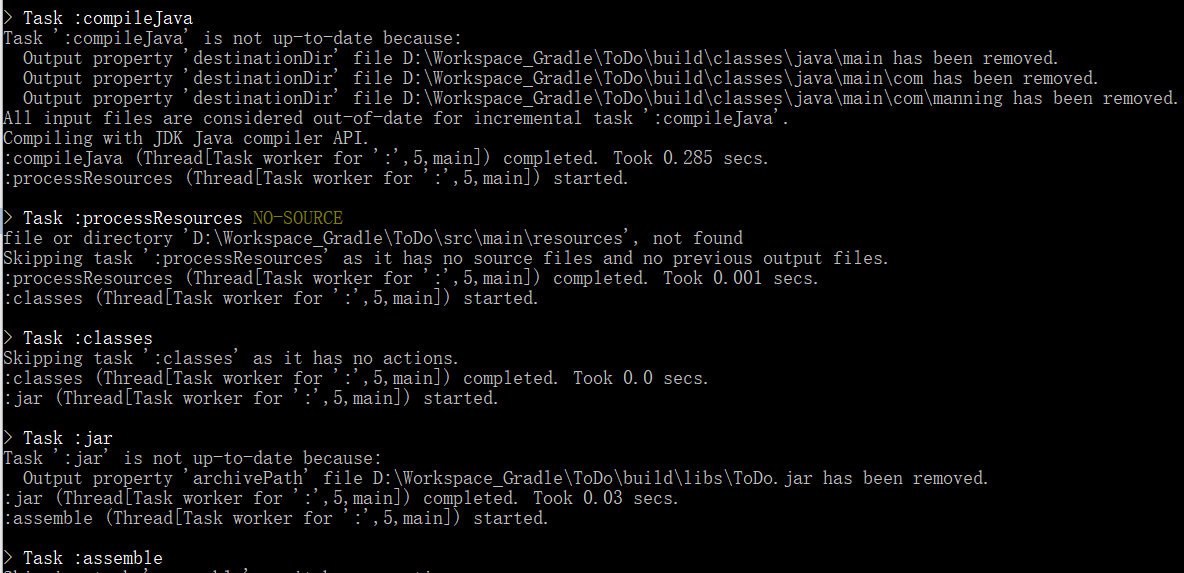
apply plugin: 'java'

一行代码足够构建你的Java代码，但是Gradle怎么知道去哪里找源文件呢？Java插件引入的约定之一就是源代码的位置。在默认情况下，插件会到src/main/java目录下查找，你需要将所有关于To Do应用程序的类文件放在正确的目录下。可以插件<https://github.com/townsfolk/gradle-templates> 中来自动生成标准约定结构的Java项目。

Java插件有一个任务叫作build，这个任务会以正确的顺序编译你的源代码，运行测试，组装jar文件，如下：



要想知道执行这个任务时都执行了哪些依赖任务，可执行：gradle build -i



某些任务标记为UP-TO-DATE消息，这意味着这个任务被跳过了，Gradle的增量式构建支持自动鉴别不需要被运行的任务。特别是在大型企业级项目中，这个特性是节省时间的好帮手。在输出中，你可以看到被跳过的任务是：comipleTestJava和testClasses。因为我们没有在src/test/java下提供单元测试。

在项目根目录下会看到生成了一个build目录里面包含了构建运行的所有输出，包括class文件、测试报告和jar文件。进入到\build\classes\java\main执行：java com.manning.gia.todo.ToDoApp即可运行程序。

对独立运行Java程序的支持：Gradle能够进一步简化构建一个独立运行的Java程序。有一个标准的Gradle扩展值得一提，它就是application插件，该插件提供了能够绑定和简化运行应用的任务。

#### 3.2.2 定制你的项目

Java插件是一个灵活的框架。它会给项目的许多方面设置有意义的默认值，比如项目结构，同时Gradle也给予你定制这些约定的能力，如何知道什么是可配置的？可查看Gradle构建语言指导：<http://www.gradle.org/docs/current/dsl/>

**修改项目和插件属性**

在下面的例子中，你将给项目指定一个版本号，并且指定Java源代码的兼容性。之前是直接运行的class文件，为了能够从Jar文件中启动应用，清单文件ANIFEST.MF需要包含信息头Main-Class。下面的清单展示了如何在构建脚本中配置默认值和头属性添加到Jar文件清单中。

apply plugin: 'java'

version = 0.1

sourceCompatibility = 1.6

jar {

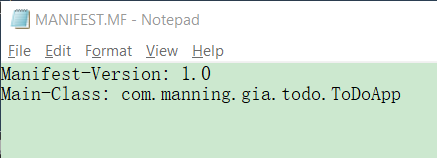
manifest {

attributes 'Main-Class': 'com.manning.gia.todo.ToDoApp'

}

}

执行gradle build命令，然后进入到\build\libs目录会看到生成了带版本号的jar，执行java -jar ToDo-0.1.jar即可启动应用。可利用winrar软件打开ToDo-01.jar，查看META-INF中的MANIFEST.MF文件，如下：



**改造遗留项目**

很少有企业级项目是在一个干净的环境下开始的。和一个遗留系统集成，迁移已有项目的技术栈，或者坚持内部标准或者限制，实在是太常见了。构建工具必须足够灵活，可以通过改变默认配置来适应来自外部的限制。假设你需要把源代码放在src目录下，而不是src/main/java，另外，你需要把结果放在out目录下，而不是build目录下，请看下面的例子：

apply plugin: 'java'

version = 0.1

sourceCompatibility = 1.6

jar {

manifest {

attributes 'Main-Class': 'com.manning.gia.todo.ToDoApp'

}

}

sourceSets {

main {

java {

srcDirs = ['src']

}

}

test {

java {

srcDirs = ['test']

}

}

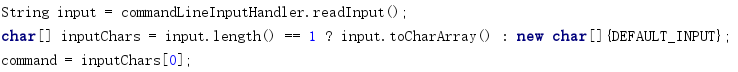
}

buildDir = 'out'

执行gradle build查看结果。定制一个构建的关键就是了解潜在的属性和DSL元素。

#### 3.2.3 配置和使用外部依赖

ToDoApp类中的main方法，通过读取用户在控制台的输入，然后将输入的第一个字符做为命令，如下：



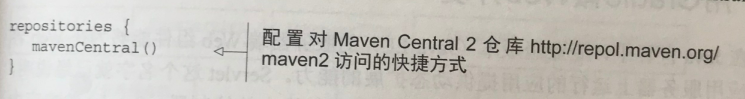
获取字符串的第1个字符的功能可以通过使用外部库来实现。最好的选择就是Apache Commons Lang库的CharUtils类。它提供了一个叫作toChar的方法可以仅仅获取String的第一个字符并转换为char返回，或者如果是一个空值则返回一个默认字符，下面的代码显示了使用外部库来获取输入的第一个字符：



那么应该如何告诉Gradle引用Apache Commons Lang库？我们来看两个DSL配置元素：repositories和dependencies。

**定义仓库**

在Java世界，类库都是以Jar文件的形式发布和使用的。许多类库都可以在仓库中找到，仓库可以是一个文件系统或者一个中心服务器，Gradle要求你定义至少一个仓库来使用依赖的类库。为此，你需要使用公共的可访问的仓库Maven Central：



定义好仓库之后，你就可以声明类库了。

**定义依赖**

一个依赖是通过group标识符、名字和一个指定版本来确定的，如下：

dependencies {

compile group: 'org.apache.commons', name: 'commons-lang3', version: '3.1'

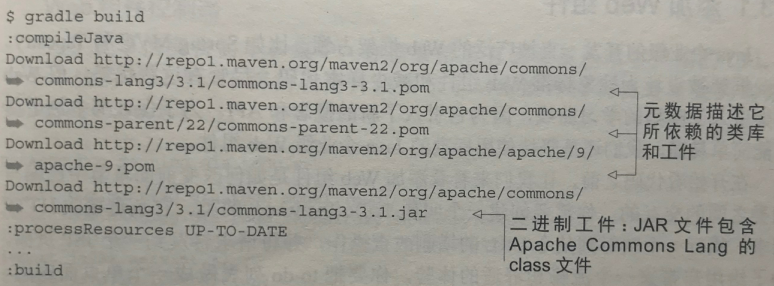
}

在Gradle中，依赖是由configuration分组的。Java插件引入的一种configuration是compile。你可以通过configuration的名字看出它是给编译源代码使用的。

如何查找一个依赖？ 在Maven Central中查找依赖的详细信息是最直接的，在<http://search.maven.org/>上，仓库提供了一个使用简单的搜索入口。

**解析依赖**

Gradle会自动检测到一个新的依赖添加到项目中。如果依赖没有被成功解析，那么就会在下个需要使用该依赖的任务启动时去下载它——在这个例子中就是compileJava任务：



### 3.3 用Gradle做Web开发

#### 3.3.1 添加Web组件

在todo包下创建一个web包，从源代码中复制ToDoServlet.java过来。

#### 3.3.2 使用War和Jetty插件

Gradle对构建和运行Web应用都提供了扩展性支持。

War插件扩展自Java插件，为Web应用部署和组装War包添加了约定与支持。在本地机器上运行一个Web应用程序是很简单的，启动快速应用部署（RAD），提供快速启动时间。在理想情况下，它不会要求你安装一个Web容器运行时环境。

Jetty是一个流行的轻量级开源Web容器，它支持所有的这些特性。它通过将一个HTTP模块添加到应用中来提供一个嵌入式实现。Gradle的Jetty插件扩展了War插件，为部署一个Web应用到嵌入式容器和运行Web应用提供了对应的任务。

**可选的嵌入式容器插件**

Jetty插件非常适合于本地Web应用开发，然而，你可能在产品环境中使用不同的Servlet容器。为了更早的在软件开发周期中提供最大的运行在不同运行时环境的能力，你需要提供其他的可用嵌入式容器实现，一个不错的选择是第三方Tomcat插件。

**War插件**

War插件扩展自Java插件，因此，只要使用了War插件则不需要使用Java插件（不过即使你也使用了Java插件也不会对你的项目带来其他影响）。应用插件是一个幂等操作，因为某一个指定的插件只会执行一次。在build.gradle文件中使用War插件，如下：

apply plugin:’war’

使用了这个插件后，你的项目除了Java插件提供的约定外，也会有Web应用的约定，它会知道Web应用文件的源代码目录，并且知道如何组装一个War文件而不是Jar文件。Web应用默认约定的源代码目录是：src/main/webapp，所有的Web资源文件都放在这个目录下，结构如下：