

++ 2013 Gradle | Евгений Борисов — Power of Gradle

https://www.youtube.com/watch?v=NZJTYPLb0iE&ab_channel=JUG.ru

Сборщики можно поделить на декларативные (Maven, Gradle) и императивные (Ant).

В императивной сборке проект состоит из большого количества модулей, мы говорим как сделать. Главный недостаток императивной сборки – большой хмл.

В maven мы не говорим как, мы говорим что сделать. Он решает откуда брать зависимости, знает как и куда паковать.

Проблема мавена, большой хмл. Мавен берет все зависимости из MavenCentral

Этапы написания плагина для мавен

1. Написание Java кода
2. Обвернуть свой плагин мавен плагиномё
3. Добавить свой плагин в тег билд

`<build>`



`</build>`

Многие вещи в мавене сложно автоматизировать, поэтому часто все выполняется руками.

Чем gradle хорош?

- Нет привязки к конкретному типу проекта
- Мощный DSL, которые легко расширять
- Groovy
- Эффективность билда
 - задачи могут бежать параллельно
 - Инкрементальная сборка

В Gradle отсутствует жесткая привязка к языку проекта. Maven заточен под Java.

Для работы gradle требуется создать файл build.gradle
В нем указывается язык

*apply **plugin**: 'java'*

В директории .gradle хранятся зависимости также как в .m2 maven

Java plugin - dependency configurations		
Name	Extends	Used by tasks
compile	-	compileJava
runtime	compile	-
testCompile	compile	compileTestJava
testRuntime	runtime, testCompile	test
archives	-	uploadArchives
default	runtime	-

В gradle есть блок configurations где можно унаследовать какую-либо конфигурацию, либо сделать свою.

Каждый раз при добавлении Dependency, она добавляется на конкретную конфигурацию.

Когда прописываются задачи, в задаче указывается с какой

конфигурацией он должен бежать.

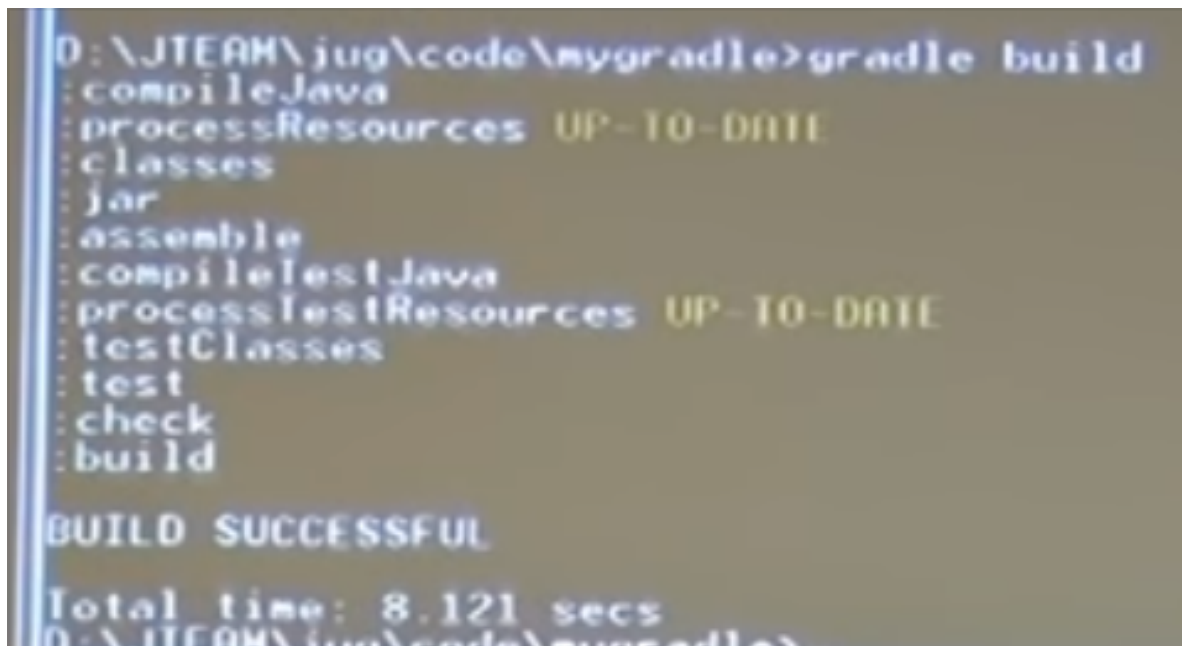
Task compileJava берет classPath из конфигурации compile.

Gradle не знает откуда подтягивать зависимости. Это указывается в пункте repositories

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

Напишем task для Junit и добавим в Gradle

```
dependencies {  
    testCompile 'junit:junit:4.11'  
}
```



```
D:\JTEAM\jug\code\mygradle>gradle build  
:compileJava  
:processResources UP-TO-DATE  
:classes  
:jar  
:assemble  
:compileTestJava  
:processTestResources UP-TO-DATE  
:testClasses  
:test  
:check  
:build  
  
BUILD SUCCESSFUL  
  
Total time: 8.121 secs  
D:\JTEAM\jug\code\mygradle>
```

Почему в консоли не выводится результат теста? Gradle выполняется в своем jvm, все внешние стримы он по умолчанию игнорирует. Каждый task является объектом и его можно настраивать, конфигурировать и переопределять.

Для того чтобы результат тестов выводился на экран необходимо переопределить объект test.

У этого объекта есть пропепти showStandardStrams, устанавливаем его true.

```

1 apply plugin : 'java'
2
3 test {
4     testLogging.showStandardStreams = true
5 }
6
7
8 repositories{
9     mavenCentral()
10 }
11
12 dependencies{
13     testCompile 'junit:junit:4.11'
14 }

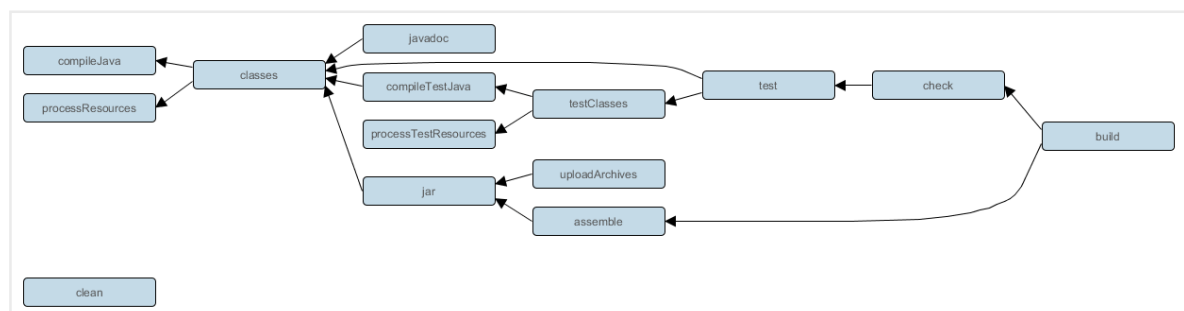
```

Для того чтобы изменения в конфигурации применились необходимо чтобы в коде что-то изменилось. Если поменять конфигурацию, но оставить код таким же, то пересборки не произойдет пока не будет вызван метод `clean` или не будут добавлены изменения в коде.

В Maven есть 3 lifecycle

1. Clean (pre-clean, clean и post-clean)
2. Site
3. Default Lifecycle (20 фаз, частично повторяющие указанные ниже фазы)

Gradle lifecycle



Lifecycle в gradle очень гибкий.

Для примера уберем зависимость build от check.

```

test {
    testLogging.showStandardStreams = true
}
build.dependsOn.remove('check')

repositories {
    mavenCentral()
}

dependencies {
    testCompile 'junit:junit:4.11'
}

```

Можно заметить что пропали 5 фаз, зависящих от check

```

Total time: 11.949 secs
D:\JTEAM\jug\code\mygradle>gradle build
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:assemble UP-TO-DATE
:build UP-TO-DATE
BUILD SUCCESSFUL

```

Напишем имитацию интеграционного теста и включим его в цепочку выполнения.

```

public class IntegrationTest {
    public static void main(String[] args) {
        System.out.println("42.0000000000000000");
    }
}

```

Далее необходимо подумать из какого плагина создавать таск, умеющий запускать main

Все таски являются объектами и создаются из классов. Внутри идет наследование. AbstractTask -> DefaultTask ...

Нас интересует таск JavaExec. Он имеет проперти, какой класс содержит main, который нужно запустить и второй проперти - classpath.

Напишем таск RunTest

```

1 apply plugin : 'java'
2
3 task runTest(type: JavaExec){
4     main = 'jugru.IntegrationTest'
5     classpath sourceSets.test.runtimeClasspath
6 }
7
8
9
10 test {
11     testLogging.showStandardStreams = true
12 }
13
14 build.dependsOn.remove('check')
15
16 repositories{

```

Результат выполнения задачи

```

Total time: 11.034 secs
D:\JTEAM\jug\code\mygradle>gradle runTest
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:compileTestJava
:processTestResources UP-TO-DATE
:testClasses
:runTest
42.000000000000000
BUILD SUCCESSFUL

```

Встроим его в цепочку к build.

Поправим gradle, вернем обратно метод check, укажем после чего будет идти метод RunTest.

```

1 apply plugin : 'java'
2
3 task runTest(type: JavaExec, dependsOn: test){
4     main = 'jugru.IntegrationTest'
5     classpath sourceSets.test.runtimeClasspath
6 }
7
8 check.dependsOn.add(runTest)
9
10 test {
11     testLogging.showStandardStreams = true
12 }
13
14
15 repositories{
16     mavenCentral()
17 }

```

Результат выполнения


```

Total time: 8.345 secs
D:\JTEAM\jug\code\mygradle>
D:\JTEAM\jug\code\mygradle>gradle build
:compileJava UP-TO-DATE
:processResources UP-TO-DATE
:classes UP-TO-DATE
:jar UP-TO-DATE
:assemble UP-TO-DATE
:compileTestJava UP-TO-DATE
:processTestResources UP-TO-DATE
:testClasses UP-TO-DATE
:test
jugru.TestService > testMsg STANDARD_OUT
    MESSAGE 2 B 11 ! 2 B
:runTest
42.0000000000000000
:check
:build

BUILD SUCCESSFUL

```

До этого мы наследовали задачи от уже имеющихся.
Как написать свой тип задач?

Давайте напишем свой Task

1. Class MyTask extends DefaultTask
2. Прописать метод @TaskAction
3. Добавить его в DSL
4. Всё!

Как получить из градла доступ к файлу с property?
application.properties

```

1 x = Leningrad gorod malenkiy

```

Можно просто вызывать переменные из файла с properties

```

1 apply plugin : 'java'
2
3 task runTest(type: JavaExec, dependsOn: test){
4     main = 'jugru.IntegrationTest'
5     classpath sourceSets.test.runtimeClasspath
6 }
7
8 task printProsto{
9     print x
10 }
11
12 check.dependsOn.add(runTest)
13
14 test {
15     testLogging.showStandardStreams = true
16 }

```

Есть несколько вариантов как сделать из Java/Groovy класса плагин для Gradle.

1. Сделать в корневой директории внутренний проект. В нем пишутся классы, связываются друг с другом, аннотируем аннотацией @Action метод который является основным.

```

import org.gradle.api.tasks.*;

public class MailTask extends DefaultTask{
    private String to;

    @Input
    public String getTo() {
        return to;
    }

    public void setTo(String to) {
        this.to = to;
    }

    @TaskAction
    public void sendMail() throws Exception{

```

У внутреннего проекта есть свой build.gradle, в котором нужно также указывать зависимости

Добавим созданный task в основной build.gradle


```

task runTest(type: JavaExec, dependsOn: test){
    main = 'jugru.IntegrationTest'
    classpath sourceSets.test.runtimeClasspath
}

task sendMail(type: mail.MailTask){
    to = recipientMail
}

check.dependsOn.add(runTest)

test {
    testLogging.showStandardStreams = true
}

```

Результат выполнения

```

* Try:
Run with --stacktrace option to get the stack trace. Run with --info or --debug
option to get more log output.
BUILD FAILED
Total time: 24.615 secs
D:\JTEAM\jug\code\mygradle>gradle sM
buildSrc:compileJava UP-TO-DATE
buildSrc:compileGroovy UP-TO-DATE
buildSrc:processResources UP-TO-DATE
buildSrc:classes UP-TO-DATE
buildSrc:jar UP-TO-DATE
buildSrc:assemble UP-TO-DATE
buildSrc:compileTestJava UP-TO-DATE
buildSrc:compileTestGroovy UP-TO-DATE
buildSrc:processTestResources UP-TO-DATE
buildSrc:testClasses UP-TO-DATE
buildSrc:test UP-TO-DATE
buildSrc:check UP-TO-DATE
buildSrc:build UP-TO-DATE
sendMail
sending mail... besvgeny@gmail.com
Building > :sendMail

```

Как сделать плагин?

Если нужно для использования в одном проекте, то можно просто сделать таск. Если все таки нужен плагин, то

- Как писать плагин:
<https://bitbucket.org/davidmc24/gradle-bintray-plugin/src/6adc1aca5ed712b8802fe6b3830b2d860d9fda88/Bintray.gradle?at=default>
- Добавить:
buildscript: apply from: 'gradle/Bintray.gradle'
- Юзать:
repositories { bintray.jcenter() }

На gradle легко мигрировать из maven и из ant.