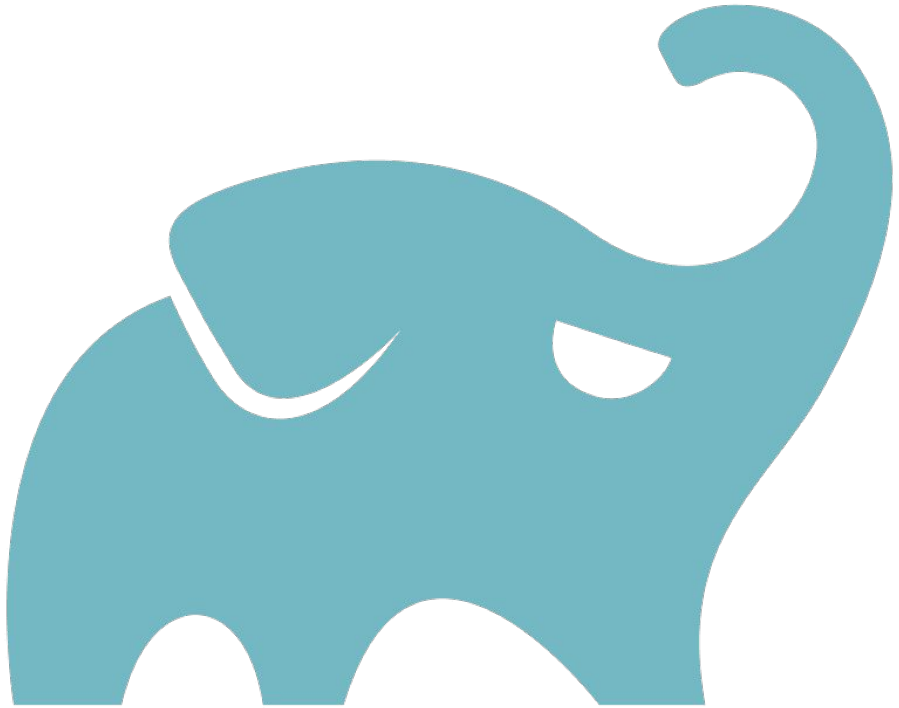
Gradle
Многомодульность

План лекции

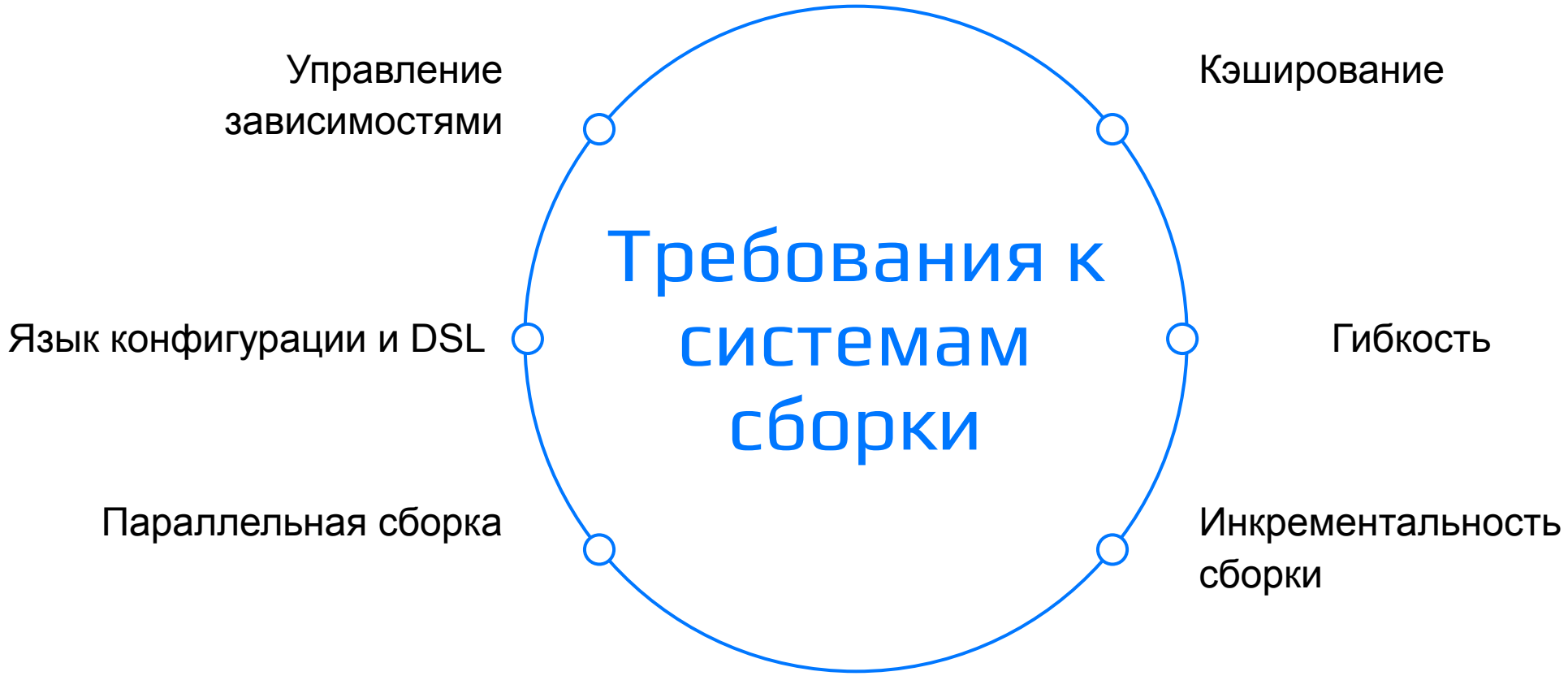
- Системы сборки
- Основные концепции Gradle
- Как управлять зависимостями с Gradle
- Как Gradle запускается
- Android Gradle Plugin
- Многомодульность
- Профайлинг

Система сборки

Инструмент для автоматизации
и оптимизации сборки

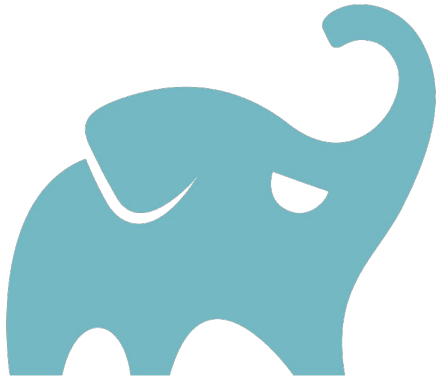


Gradle

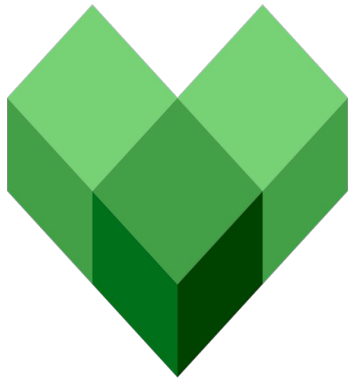


https://threadreaderapp.com/thread/1466134453940629515.html

Альтернативы



Gradle



Bazel (Blaze) by Google

Используется для сборки AOSP

Buck by Facebook (Meta)

Делался выходцами из Google

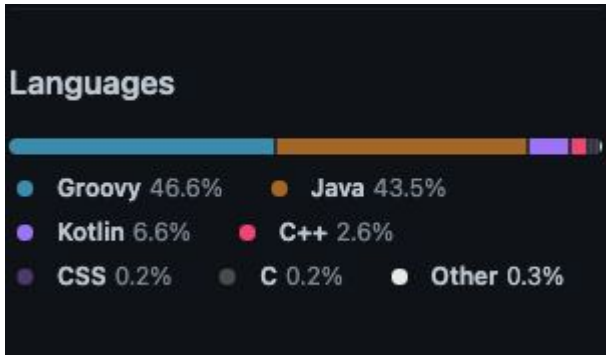
1. https://www.bruceeckel.com/2021/01/02/the-problem-with-gradle/
2. https://melix.github.io/blog/2021/01/the-problem-with-gradle.html
3. https://blog.gradle.org/gradle-vs-bazel-jvm
4. https://developer.squareup.com/blog/stampeding-elephants/

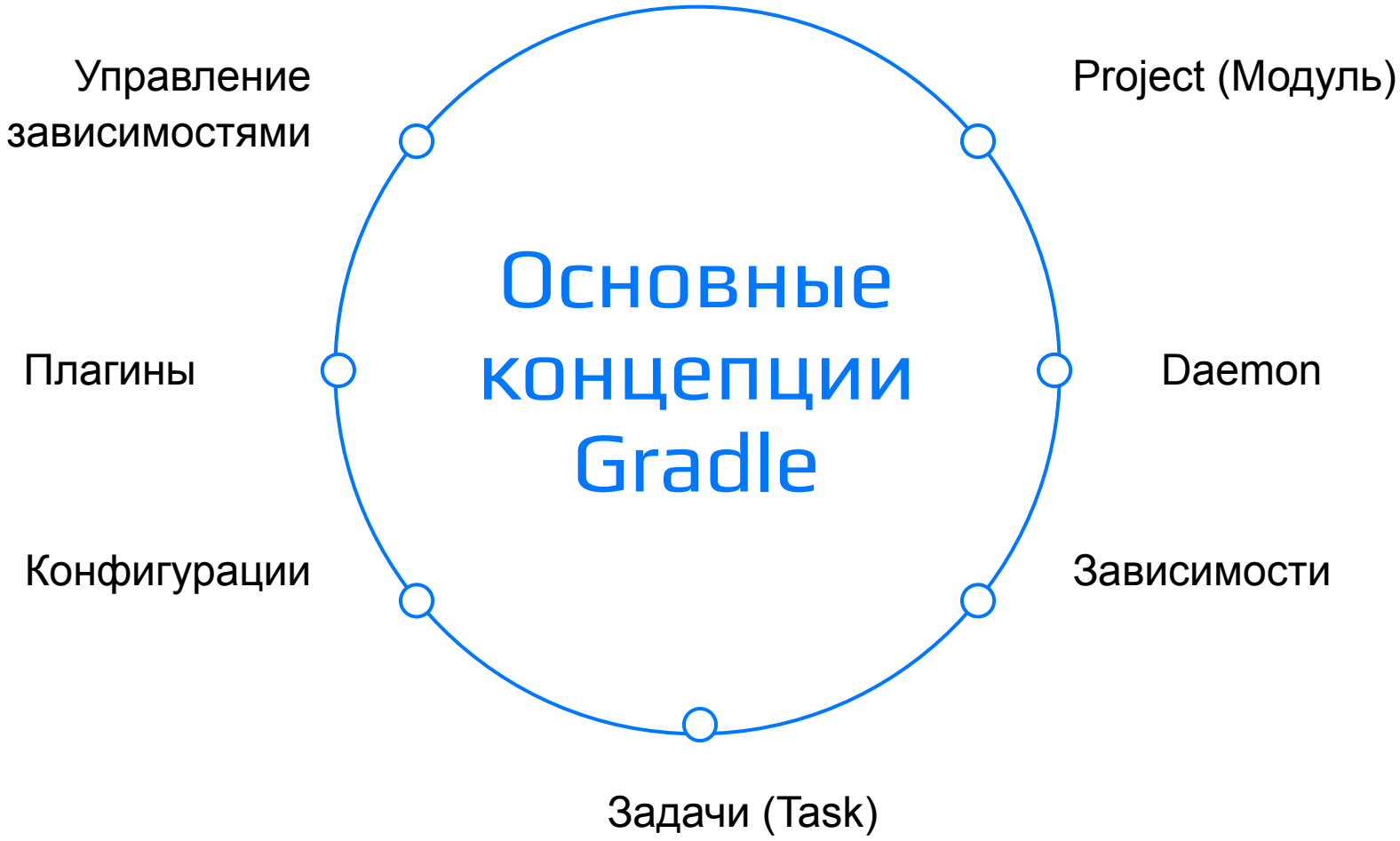
Почему стоит изучать Gradle?

- Не понимаете почему вы используете Gradle и почему он используется для сборки приложений в Android Studio?
- У вас возникает чувство, что ненавидите Gradle?
- Вы не понимаете, что происходит в Gradle и что это строки, которые печатаются в консоли во время сборки?
- Собираете проект всегда надеясь на удачу, что он соберется?
- Хотите стать специалистом в Android разработке?
- Хочется, чтобы все собиралось быстрее, но сначала надо починить сборку?
- Хотите понять о какой там многомодульности все говорят?
- Вкатывание в KMM удерживает необходимость настройки проекта?

Основные характеристики Gradle

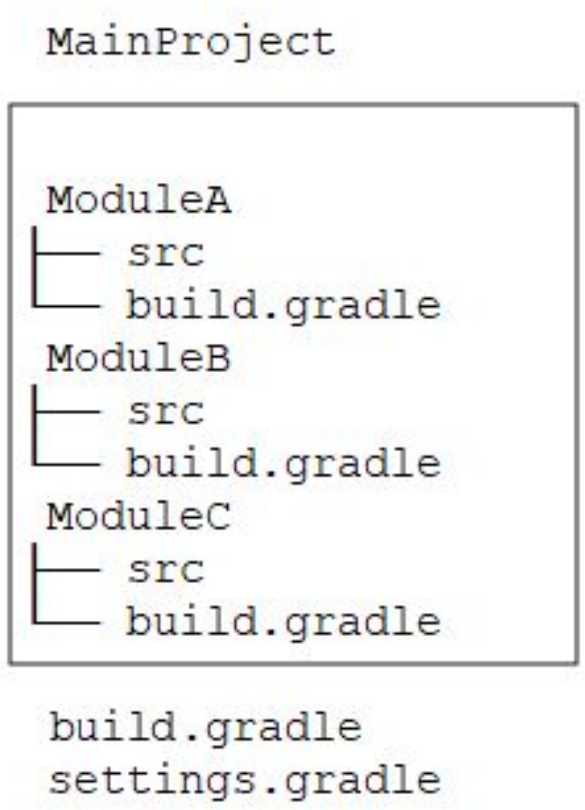
- Проект с открытым исходным кодом
- Основан на JVM. Если посмотреть на исходники, то можно увидеть, что большая часть кода написана на JVM языках.
- Скрипты можно писать на Kotlin и Groovy. Для его разработки используются обычно JVM языки, такие как Kotlin, Java и Groovy. С помощью этих языков можно и расширять функциональность Gradle.
- Предоставляет возможность для расширения функционала через плагины
- Является официальный инструментом сборки Android приложений




Управление зависимостями
Project (Модуль)
Основные концепции Gradle
Плагины
Daemon
Конфигурации
Зависимости
Задачи (Task)

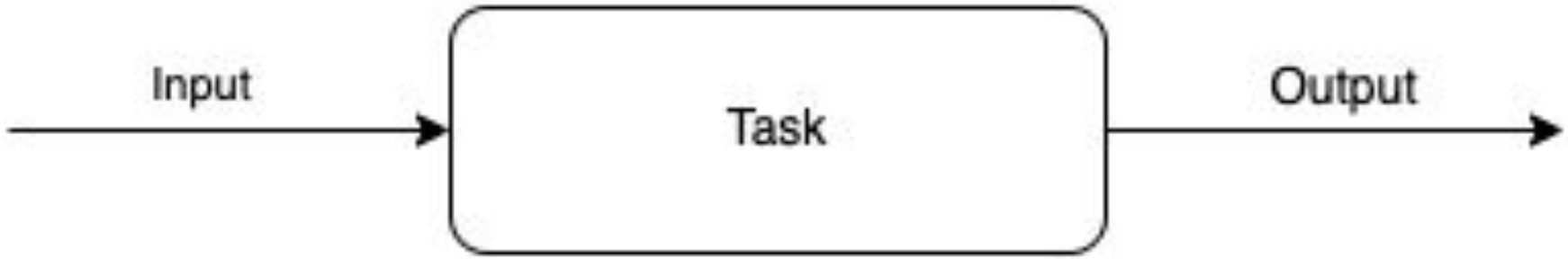
Gradle Project

- В привычном понимании это модуль
- В коде представляет из себя экземпляр класса Project
- Может быть рутовый проект и много дочерних
- Конкретный проект настраивается в **build.gradle.kts**
- Для рутового проекта, который включает в себя множество других проектов настраивается **settings.gradle.kts**
- Для проекта существуют файлы с переменными **gradle.properties** и **local.properties** для конфигурации этого проекта



Gradle Task

- Выполняют атомарную работу
- Большинство нужных задач уже есть
- При необходимости мы можем добавить свой



Gradle Dependencies

- Объявляются в блоке **dependencies**
- Есть возможность настраивать через **dependencyResolutionManagement**
- При объявлении состоит их конфигурации и самой зависимости
- Зависимостью может являться файл, директория, проект или библиотека

```
dependencies { this: DependencyHandlerScope
    implementation(project("moduleA"))
    implementation(platform("androidx.compose:compose-bom:2022.10.00"))
    implementation( dependencyNotation: "androidx.compose.ui:ui")
}
```

Gradle Configurations

- Определяют скоуп зависимости, где она может быть использована
- Существует трех видов: bucket, resolvable и consumable
- Уже есть определенные конфигурации, но можно еще создать свои

```
dependencies { this: DependencyHandlerScope
    implementation(project("moduleA"))
    implementation(platform("androidx.compose:compose-bom:2022.10.00"))
    implementation( dependencyNotation: "androidx.compose.ui:ui")
}
```

https://docs.gradle.org/current/userguide/declaring_dependencies.html#sec:what-are-dependency-configurations

Gradle Plugin

Используется для расширения функционала

Обычно в них добавляется:

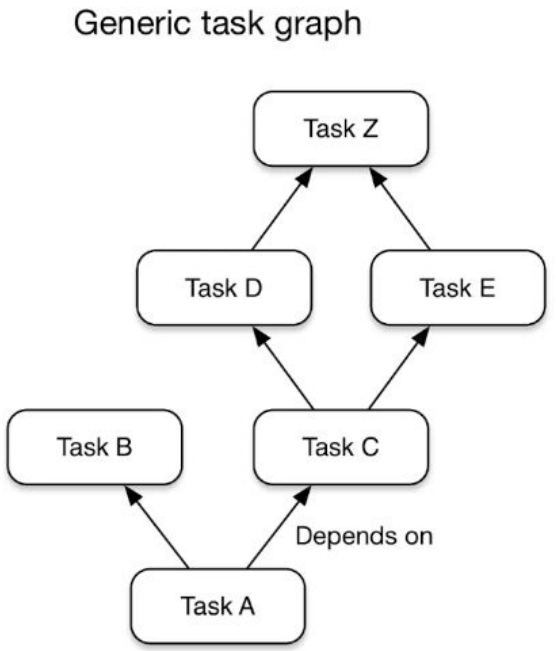
- таски
- конфигурации
- настройка проекта
- Регистрация extensions для DSL

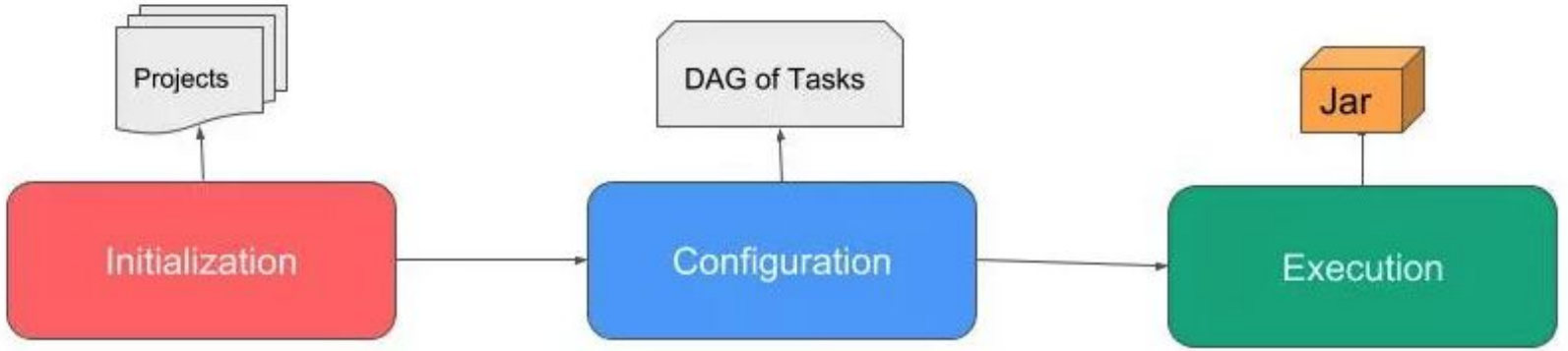
Используемые в Android разработке плагины:

- Android Gradle Plugin
- Java Plugin
- Kotlin Plugin
- Ksp
- Kapt
- kotlin Serialization
- plugins.gradle.org

Gradle Lifecycle

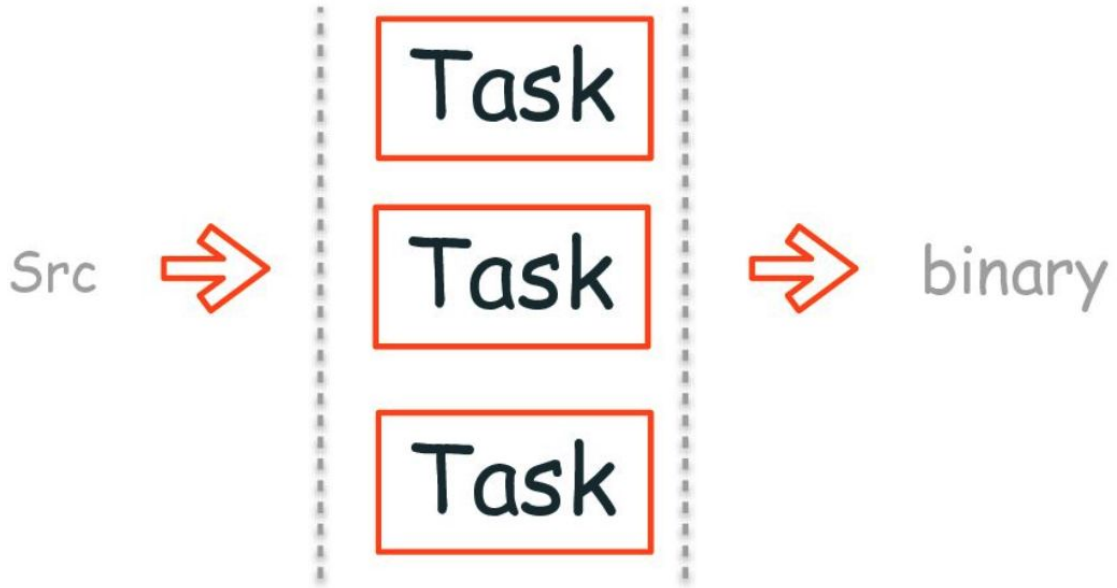
- **Initialization** - билдит settings.gradle.kts
- **Configuration** - строит граф задач и проектов, подкладывает properties. выполняет build.gradle.kts
- **Execution** - выполняет задачи согласно графу в Gradle Daemon





Code simple task

Способы выполнения Gradle Task



Зависимости между Gradle Task

- dependsOn (влияет на запуск и порядок тасок)
- Неявная зависимость тасок через input и output (Является ленивой)
- finalizeBy (влияет на запуск и порядок тасок)
- shouldRunAfter (влияет на порядок тасок)
- mustRunAfter (влияет на порядок тасок)

Зависимости между Gradle Task

- **dependsOn (влияет на запуск и порядок тасок)**
  - **таска, которая должна выполниться перед текущей**
- Неявная зависимость тасок через input и output
- finalizeBy (влияет на запуск и порядок тасок)
- shouldRunAfter (влияет на порядок тасок)
- mustRunAfter (влияет на порядок тасок)

```
tasks.register( name: "check") { this: Task
    dependsOn( ...paths: "assemble")
}
```

Зависимости между Gradle Task

- dependsOn (влияет на запуск и порядок тасок)
- **Неявная зависимость тасок через input и output**
  - **Является ленивой в отличие от dependsOn**
- finalizeBy (влияет на запуск и порядок тасок)
- shouldRunAfter (влияет на порядок тасок)
- mustRunAfter (влияет на порядок тасок)

```
val producer = tasks.register<Producer>("producer")
val consumer = tasks.register<Consumer>("consumer")

consumer {
    // Connect the producer task output to the consumer task input
    // Don't need to add a task dependency to the consumer task. This is automatically added
    inputFile.set(producer.flatMap { it.outputFile })
}

producer {
    // Set values for the producer lazily
    // Don't need to update the consumer.inputFile property. This is automatically updated as producer.outputFile changes
    outputFile.set(layout.buildDirectory.file("file.txt"))
}
```

Зависимости между Gradle Task

- dependsOn (влияет на запуск и порядок тасок)
- Неявная зависимость тасок через input и output (Является ленивой)
- **finalizeBy (влияет на запуск и порядок тасок)**
  - **Определяет какая таска должна быть выполнена после текущей. Выполнится даже если та не завершится успешно**
- shouldRunAfter (влияет на порядок тасок)
- mustRunAfter (влияет на порядок тасок)

Зависимости между Gradle Task

- dependsOn (влияет на запуск и порядок тасок)
- Неявная зависимость тасок через input и output (Является ленивой)
- finalizeBy (влияет на запуск и порядок тасок)
- **shouldRunAfter (влияет на порядок тасок)**
  - **Влияет на порядок, если только они должны оба запуститься, в ином случае просто запускает таску. Порядок может не сохраниться, если есть циклическая зависимость между тасками или таски запускаются параллельно**
- mustRunAfter (влияет на порядок тасок)

Зависимости между Gradle Task

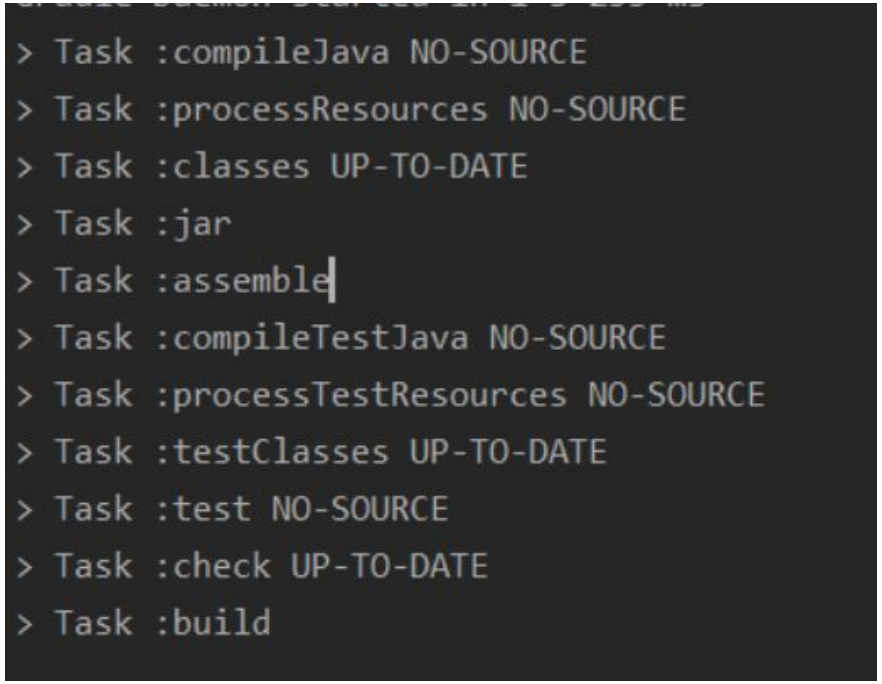
- dependsOn (влияет на запуск и порядок тасок)
- Неявная зависимость тасок через input и output (Является ленивой)
- finalizeBy (влияет на запуск и порядок тасок)
- shouldRunAfter (влияет на порядок тасок)
  - Влияет на порядок, если только они должны оба запуститься, в ином случае просто запускает таску. Порядок может не сохраниться, если есть циклическая зависимость между тасками или таски запускаются параллельно
- **mustRunAfter (влияет на порядок тасок)**
  - **Влияет на порядок Gradle Task, если только они должны оба запуститься, в ином случае просто запускает таску. Порядок тасок должен быть всегда, если обе таски планируются к запуску, в отличие от *shouldRunAfter*, который может не гарантировать порядок**

Code simple dependsOn task

Gradle Task Label

Во время билда Gradle Task могут оказаться в следующих состояниях:

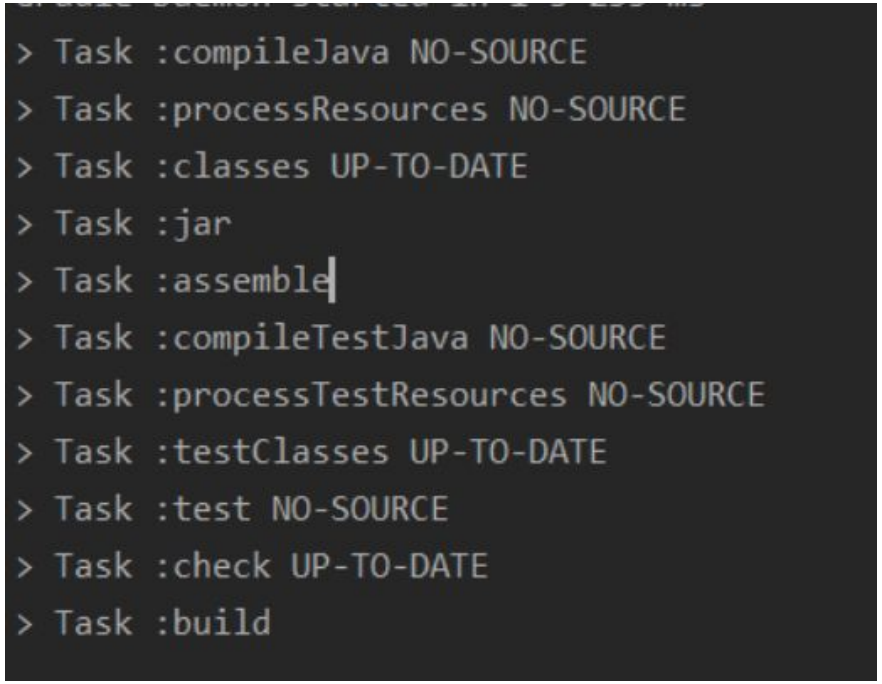
- **NO-SOURCE** - нет input, нечего выполнять
- **SKIPPED** - выключили флагом enabled или не отработал onlyIf в таске
- **UP-TO-DATE** - input не изменился. результаты актуальны и лежат в build папке (Incremental Cache). Либо у таска нету input для выполнения или она зависит от тасок, статус которых не EXECUTED
- **FROM-CACHE** - результат по input нашелся в кэше (Build Cache)
- **EXECUTED(no label)** - таска выполнилась



Gradle Task Label

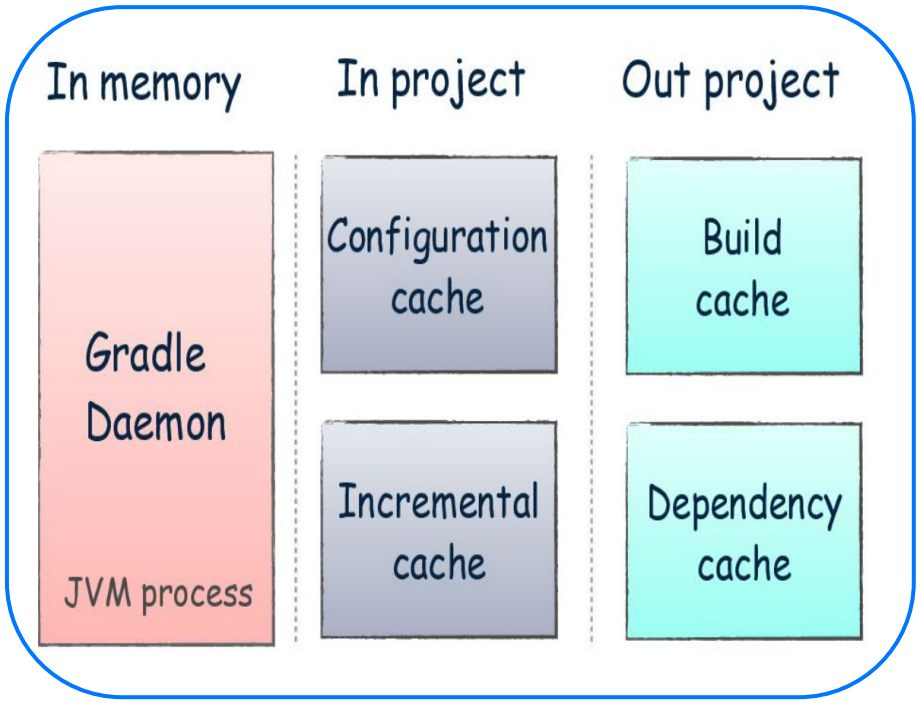
Во время билда Gradle Task могут оказаться в следующих состояниях:

- NO-SOURCE - нет input, нечего выполнять
- SKIPPED - выключили флагом enabled или не отработал onlyIf в таске
- **UP-TO-DATE** - input не изменился. результаты актуальны и лежат в build папке (Incremental Cache). Либо у таска нету input для выполнения или она зависит от тасок, статус которых не EXECUTED
- FROM-CACHE - результат по input нашелся в кэше (Build Cache)
- EXECUTED(no label) - таска выполнилась

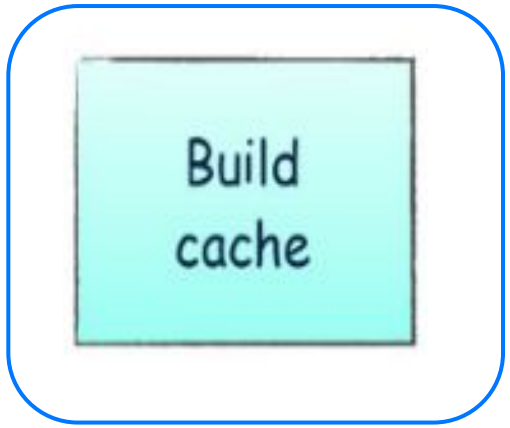


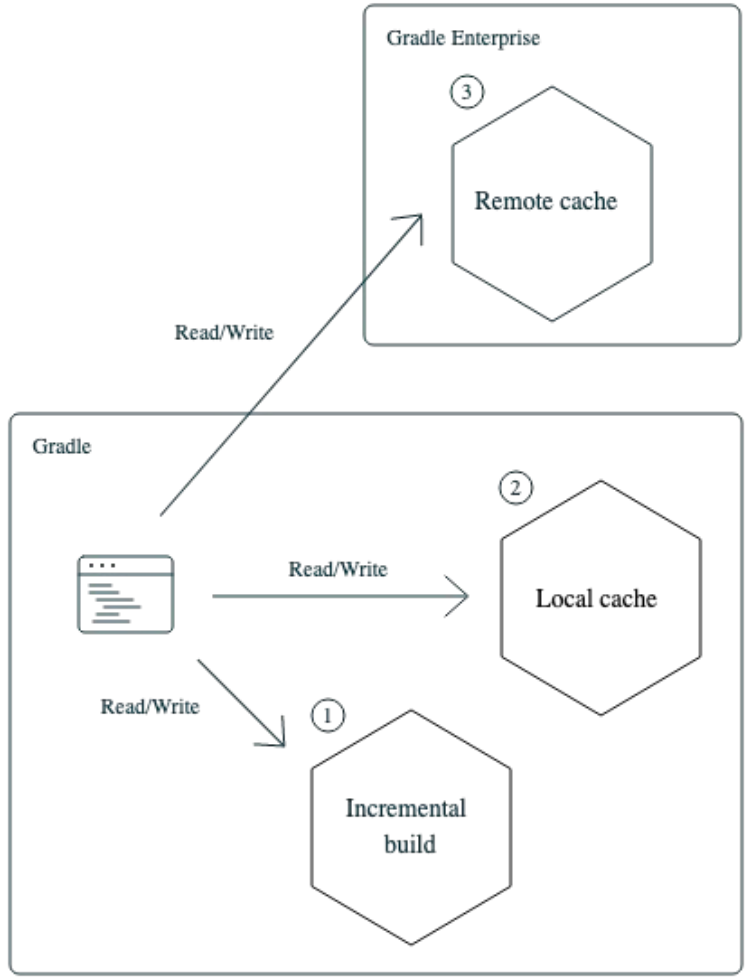
Gradle Cache

Local



Remote





Gradle Task Cache

- **Incremental** - таски, которые могут проверять насколько и как изменились input и что в этом случае выполнять
- **Cacheable** - кэшируемые таски сохраняют результат согласно input/output. Gradle сам по этим данным определит изменился ли input

Code cacheable/incremental task

Gradle Configurations. Bucket

Основные:

configuration	When available?	Leaks into consumers' compile time?	Leaks into consumers' runtime?
implementation	compile time runtime	no	yes
api	compile time runtime	yes	yes
compileOnly	compile time	no	no
runtimeOnly	runtime	no	yes

Специальные:

testImplementation
androidTestImplementation
….

https://medium.com/agorapulse-stories/gradle-configurations-explained-4b9608dd5e35

Custom Gradle Configuration

```
configurations {
    skikoNativeX64
    skikoNativeArm64
}

def jniDir = "${projectDir.absolutePath}/src/main/jniLibs"

def unzipTaskX64 = tasks.register("unzipNativeX64", Copy) {
    def skikoNativeX64 = configurations.skikoNativeX64
    from(skikoNativeX64.getFiles().collect { zipTree(it) })
    into(file("$jniDir/x86_64"))
}

def unzipTaskArm64 = tasks.register("unzipNativeArm64", Copy) {
    def skikoNativeArm64 = configurations.skikoNativeArm64
    from(skikoNativeArm64.getFiles().collect { zipTree(it) })
    into(file("$jniDir/arm64-v8a"))
}

dependencies {
    def skiko_version = "0.7.15"
    implementation "org.jetbrains.skiko:skiko-android:$skiko_version"

    skikoNativeX64("org.jetbrains.skiko:skiko-android-runtime-x64:$skiko_version")
    skikoNativeArm64("org.jetbrains.skiko:skiko-android-runtime-arm64:$skiko_version")
}

tasks.withType(org.jetbrains.kotlin.gradle.dsl.KotlinJvmCompile).configureEach {
    dependsOn(unzipTaskX64)
    dependsOn(unzipTaskArm64)
}
```

Реализация кастомных конфигурации:
https://www.youtube.com/watch?v=yDj0n0g5dXY
https://docs.gradle.org/current/userguide/declaring_dependencies.html

Репозиторий зависимостей Gradle

Два способа

- Project.repositories в build.gradle - старый способ
- dependencyResolutionManagement в settings.gradle - новый предпочтительный вариант

```
allprojects { this: Project
    repositories { this: RepositoryHandler
        google()
        jcenter()
    }
}
```

```
dependencyResolutionManagement { this: DependencyResolutionManagement
    repositoriesMode.set(RepositoriesMode.FAIL_ON_PROJECT_REPOS)
    repositories { this: RepositoryHandler
        google()
        mavenCentral()
    }
}
```

Репозиторий зависимостей Gradle

Порядок зависимостей имеет значение!

Gradle ищет сверху вниз

Через maven можно подключать свои репозитории

```
repositories { this: RepositoryHandler
    google { this: MavenArtifactRepository
        content { this: RepositoryContentDescriptor
            // ищет только артефакты с этой группой
            includeGroup( group: "androidx.compose.ui")
        }
        mavenContent { this: MavenRepositoryContentDescriptor
            // только релизные сборки
            releasesOnly()
        }
    }
    // предопределенный репозиторий
    mavenCentral()
    flatDir { this: FlatDirectoryArtifactRepository
        dirs("lib")
    }

    // репозитории, где нужен доступ
    maven { this: MavenArtifactRepository
        url = uri(...)
        credentials { this: PasswordCredentials
            username = "user"
            password = "password"
        }
    }
    // maven local, но своя директория
    maven(url = File("myMavenLocal").toURI())
    // по умолчанию лежит ~./m2 на компьютере
    mavenLocal()
}
```

https://docs.gradle.org/current/userguide/declaring_repositories.html#sub:flat_dir_resolver

Gradle Dependencies Resolution Strategy

- **failOnVersionConflict** - падать при конфликте версии
- **preferProjectModules** отдавать предпочтение зависимостям в виде модулей в проекте
- **dependencySubstitution** - позволяет указать какую зависимость на какую заменить
- **cacheChangingModulesFor** - кэшировать изменения в модулях, которые часто меняются. Нужно указать срок, по умолчанию один день
- **cacheDynamicVersionsFor** - кэшировать зависимости, версии которых часто меняются. Нужно указать срок, по умолчанию один день
- **force** - позволяет вбить гвоздями версию для какой-то зависимости. Причем гвоздями вбиваются даже версии транзитивных зависимостей
- **setForcedModules** делает то же самое, но переписывает то, что записалось с force
- **eachDependency** - итерация по зависимостям и выполнение действии над ними
- **dependencies.constraints** в рамках проекта можно определить версию зависимости, которая будет использоваться, даже если она тянется транзитивно
- **configuration.exclude** - позволяет точечно исключить некоторые транзитивные зависимости

```
configurations.all { this: Configuration
    resolutionStrategy { this: ResolutionStrategy
        failOnVersionConflict()
        preferProjectModules()
        dependencySubstitution { this: DependencySubstitutions
            substitute(project( path: ":util")).using(project( path: ":utils2"))
        }
        cacheChangingModulesFor( value: 5, TimeUnit.HOURS)
        cacheDynamicVersionsFor( value: 5, TimeUnit.HOURS)
        force( ...moduleVersionSelectorNotations: "androidx.lifecycle:lifecycle-runtime-ktx:2.5.1")
        setForcedModules("androidx.lifecycle:lifecycle-runtime-ktx:2.5.1")
        eachDependency { this: DependencyResolveDetails
            // some rule
        }
        //  ... и тд
    }
}

dependencies { this: DependencyHandlerScope

    implementation("androidx.lifecycle:lifecycle-runtime-ktx")
    constraints { this: DependencyConstraintHandlerScope
        implementation( constraintNotation: "androidx.lifecycle:lifecycle-runtime-ktx:2.5.1") { this: DependencyConstraint
            because( reason: "i want")
        }
    }

    implementation( dependencyNotation: "some:dep:1.1-alpha08") { this: ExternalModuleDependency
        exclude( group: "androidx.lifecycle",  module: "lifecycle-runtime-ktx")
    }
}
```

Gradle Plugin Types

Gradle Plugin можно разделить на два типа:

- **Script плагины** - плагин в .gradle.kts файле, который подключается через apply from
- **Бинарные плагины** - плагин в отдельном модуле или buildSrc. Тот же скрипт, либо реализация Plugin класса.

Стоит отдать предпочтение бинарным плагинам, потому что скрипты чаще всего разбросаны, меньше функционала предоставляют, могут плохо влиять на сборку и распространять их между проектами сложнее.

https://docs.gradle.org/current/userguide/plugins.html#sec:types_of_plugins

apply plugin vs plugins {}

Есть два способа подключение плагинов apply plugin или plugins {}

Ключевые отличия:

- apply plugin можно писать где угодно в build скрипте, а plugin только наверху.
- plugins {} блок может искать плагины на Gradle Plugin Portal, а для apply plugin нужно подключать classpath откуда скачивать в buildscripts.dependencies
- Блок plugins выполняется интерпретатором, который позволяет быстрее выполнять build.gradle.kts за счет кэширования результата

plugins {} наиболее рекомендуемый и развиваемый вариант

Про интерпретатор - https://docs.gradle.org/8.0/release-notes.html
Особенности, чтобы сработал интерпретатор - https://docs.gradle.org/8.0/userguide/plugins.html#sec:constrained_syntax

Репозиторий для Gradle Plugin

- **pluginManagement** в settings.gradle.kts. По аналогии с dependencyResolutionManagement и resolutionStrategy
- **buildscript** - legacy вариант

```
pluginManagement { this: PluginManagementSpec
    resolutionStrategy { this: PluginResolutionStrategy
        eachPlugin { this: PluginResolveDetails

        }
    }
    repositories { this: RepositoryHandler
        maven { this: MavenArtifactRepository
            url = uri(...)
        }
        google()
        mavenCentral()
        gradlePluginPortal()
    }
}
```

```
buildscript { this: ScriptHandlerScope
    repositories { this: RepositoryHandler
        google()
        jcenter()
    }
    dependencies { this: DependencyHandlerScope
        classpath("com.android.tools.build:gradle:3.5.4")
    }
}
```

Где писать Gradle Plugin

- **buildSrc**
  - название зарезервировано. Может быть только один buildSrc
  - Изменение приводит к инвалидации кэша
- **Composite Build**
  - Способ создания проекта, который будет использовать при сборке другого проекта. Создается отдельный Gradle проект, в котором есть settings.gradle.kts и build.gradle.kts, в котором подключаются зависимости для создания плагинов и настройки проекта.
  - Чтобы подключить такой модуль к проекту в settings.gradle.kts в pluginManagement необходимо прописать `includeBuild("./composite-module-name")
  - Использование Composite Builds в отличие от buildSrc Gradle не инвалидирует Gradle Cache и позволяет делать многомодульные проекты с плагинами, чтобы отключать те, в которых ведется активная разработка

Ссылка: https://docs.gradle.org/current/userguide/organizing_gradle_projects.html#sec:build_sources

Пример плагинов buildSrc/Composite Build

Способы хранения версии зависимостей

- extra properties
- buildSrc dependencies
- Composite Builds dependencies
- Gradle Versions Catalog

https://github.com/material-components/material-components-android/blob/master/build.gradle
https://funkymuse.dev/posts/toml-gradle/
https://docs.gradle.org/current/userguide/platforms.html

extra properties

- Удобен в groovy
- Для kts чаще всего придется накостылять
- Нет автокомплита!

Вариант для Kotlin

versions.gradle.kts

```
extra.apply {
    set("lifecycle", "2.5.1")
    set("core", "1.8.0")
}
```

root build.gradle.kts

```
buildscript {
        apply(from = "versions.gradle.kts")
}
```

build.gradle.kts

```
val lifecycle: String by rootProject.extra
val core: String by rootProject.extra

dependencies {
    implementation("androidx.core:core-ktx:$core")
    implementation("androidx.lifecycle:lifecycle-runtime-ktx:$lifecycle")
}
```

Вариант для Groovy

versions.gradle

```
ext {
    libs = [
        androidX : androidX = [
            core     : 'androidx.core:core-ktx:1.8.0',
            lifecycle: 'androidx.lifecycle:lifecycle-runtime-ktx:2.5.1',
        ]
    ]
}
```

root build.gradle.kts

```
buildscript {
        apply(from = "versions.gradle")
}
```

build.gradle.kts

```
dependencies {
    implementation libs.androidX.lifecycle
    implementation libs.androidX.core
}
```

https://github.com/material-components/material-components-android/blob/master/build.gradle

extra properties. Хотим как в Groovy

Вариант через Map

```
val libs = rootProject.ext["libs"] as Map<String, Map<String, String>>
val lifecycle: String = libs["androidX"]?.get("lifecycle").orEmpty()
val core: String = libs["androidX"]?.get("core").orEmpty()

dependencies {
    implementation(core)
    implementation(lifecycle)
}
```

Вариант через ссылку на метод

versions.gradle

```
ext {
    …
    resolveDep = this.&resolveDep
}
def resolveDep(name) {
    switch (name) {
        case "core":
            return libs.androidX.core
        case "lifecycle":
            return libs.androidX.lifecycle
        default:
            throw new IllegalArgumentException("No mapping exists for
name: $name.")
    }}
```

build.gradle.kts:

```
val resolveDep: groovy.lang.Closure<Any> by rootProject.ext
dependencies {
    implementation(resolveDep("core"))
    implementation(resolveDep("lifecycle"))
}
```

Composite Build dependencies

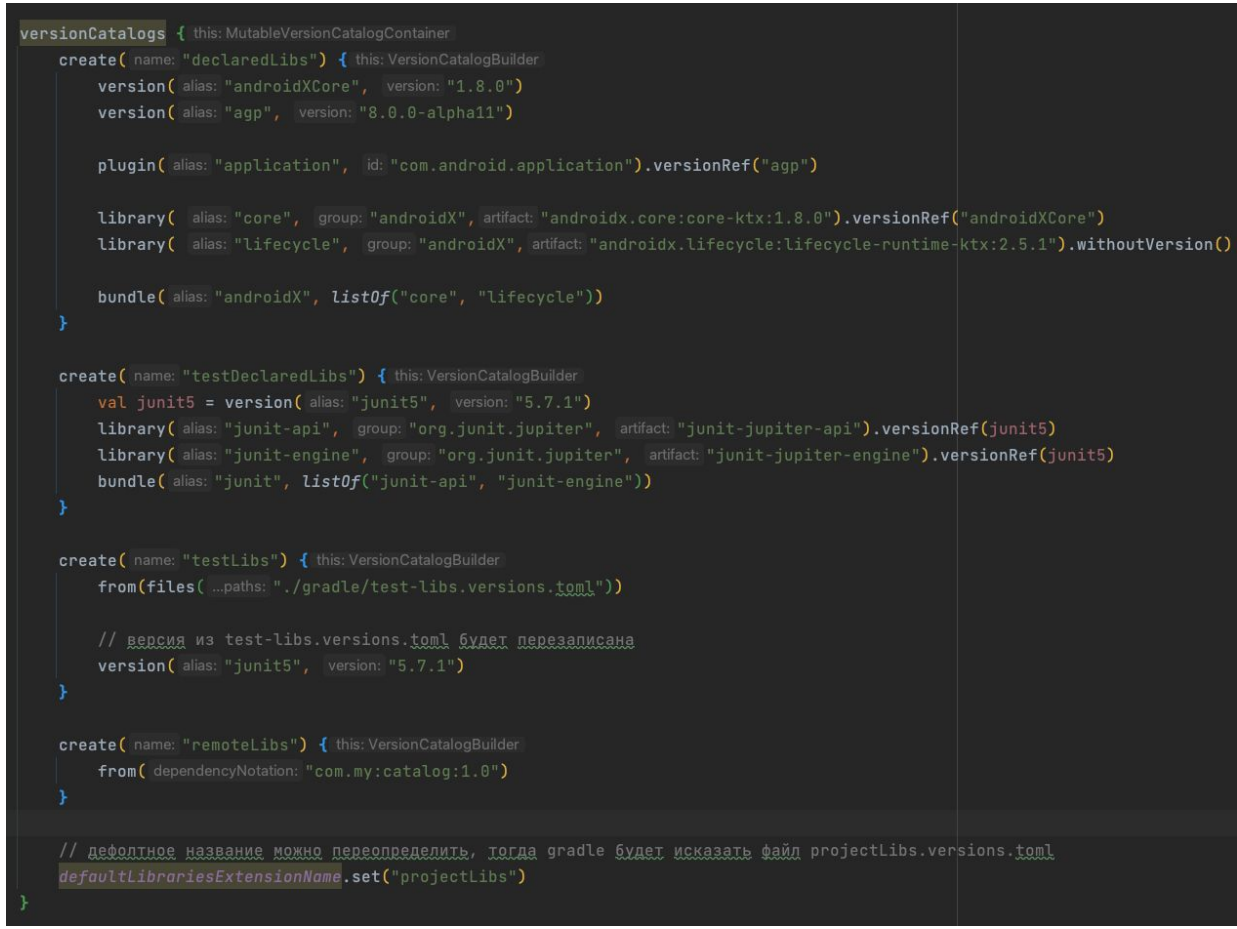
- Аналогично buildSrc
- Изменение не инвалидирует Gradle Cache
- Чтобы иметь возможность использовать классы необходимо подключить хотя бы один плагин к модулю

Gradle Versions Catalog

- Объявление в блоке versionsCatalogs в dependencyResolutionManagement в settings.gradle.kts. Удобно?
- Можно прописывать зависимости сразу, а можно унести в toml файл
- toml с названием **libs.versions.toml** задекларирован по умолчанию в папке ./gradle, но можно переопределить название
- есть возможность добавлять свои toml файлы

Прописывать можно:

- version
- библиотеки и их группы, версии. есть возможность настраивать правила для версии
- bundle - набор библиотек, которые можно подключать как одна зависимость
- plugins



https://funkymuse.dev/posts/toml-gradle/
https://docs.gradle.org/current/userguide/platforms.html

TOML

Пример файла

```toml
[versions]
androidXCore = "1.8.0"
agp = "8.0.0-alpha11"

[libraries]
#androidCore = { group = "androidx.core", name = "core-ktx", version.ref = "androidXCore" }
#androidLifecycle = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version = "2.5.1" }

#Если создавать alias через дефис, то будет сгенерирован доп раздел
android-core = { group = "androidx.core", name = "core-ktx", version.ref = "androidXCore" }
android-lifecycle = { group = "androidx.lifecycle", name = "lifecycle-runtime-ktx", version = "2.5.1" }

[bundles]
androidX = ["android-core", "android-lifecycle"]

[plugins]
application = { id = "com.android.application", version.ref = "agp" }
```

Пример использования

```
plugins { this: PluginDependenciesSpecScope
    kotlin("android")
    alias(libs.plugins.application)
}
```

```
// libraries
implementation(libs.android.lifecycle)
implementation(libs.android.core)
testImplementation(testLibs.junit.api)
testImplementation(testLibs.junit.engine)

// bundle
implementation(libs.bundles.androidX)
testImplementation(testLibs.bundles.junit)
```

Idea подсвечивает плагины красным как ошибка:
Проблема - https://youtrack.jetbrains.com/issue/KTIJ-19369
Фикс - https://github.com/gradle/gradle/issues/22797
Почему работало - https://github.com/JetBrains/kotlin/blob/3d65420f78e384d2e4f1eeb49b935696a50f0bff/libraries/scripting/common/src/kotlin/script/experimental/api/scriptEvaluation.kt#L56-L60

Gradle Daemon

Для чего он нужен?

- хранит in memory кэш
- держит JVM всегда на готове
- работают оптимизации JVM, которые позволяют прогрессивно оптимизировать выполнение сборки
- следит за файловой системой

Но есть проблемы?

Их может быть больше не нужно - Помогает ./gradlew --stop

Излишняя прожорливость - можно настраивать в gradle.properties, но есть один случай…

Способы запуска Gradle

Для чего он нужен?

- В командной строке выполнить команду gradle someTask
- В командной строке в директории проекта выполнить команду ./gradlew someTask
- Кликнуть кнопку в Android Studio для синка проекта или запуска

gradle someTask

$ /path/to/local/distribution/bin/gradle SomeTask

Shell process

Executes the script…

Search and find correct **`java`** command.

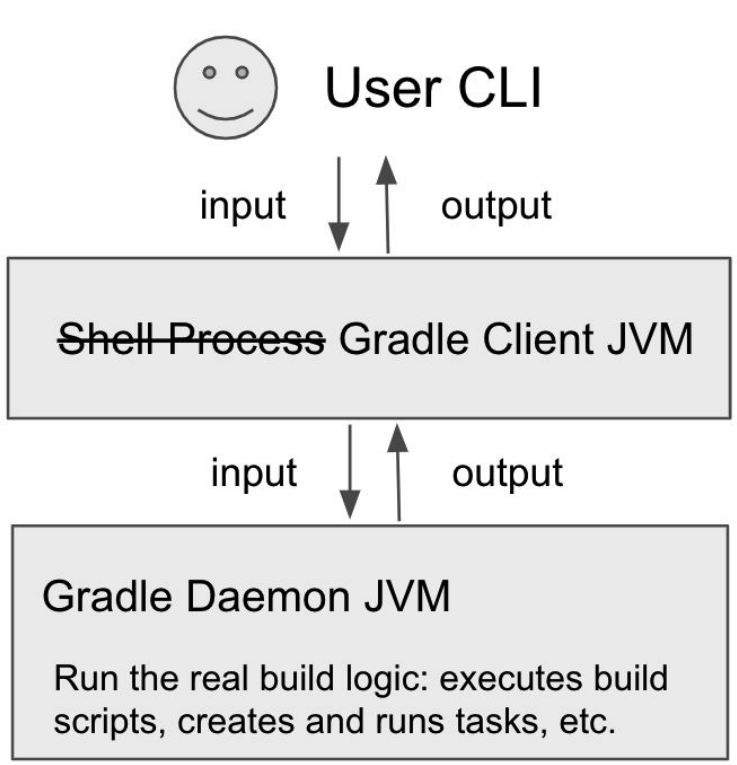
Determine the parameters passed to **`java`** command.

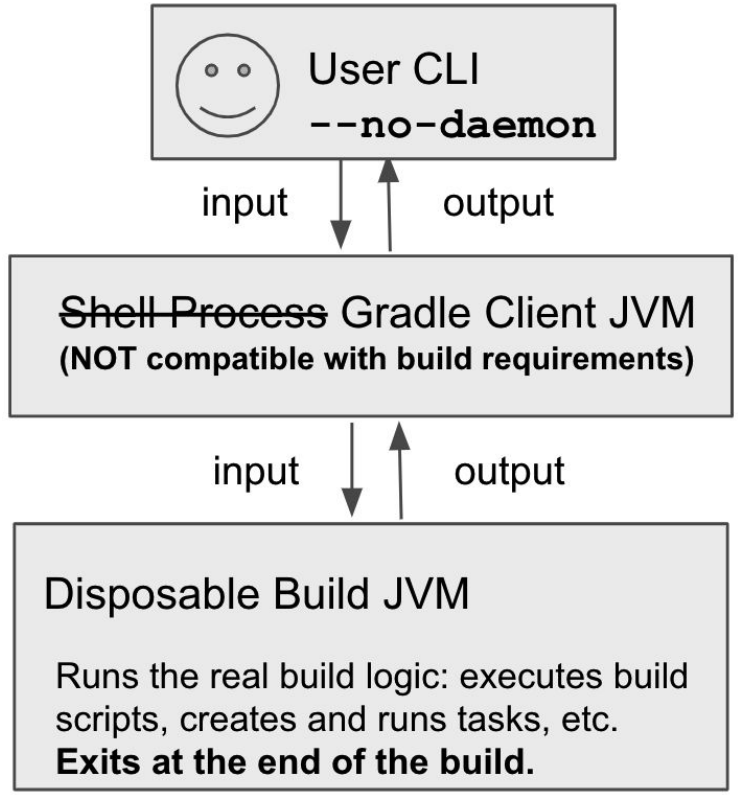
```
exec /path/to/java -Xmx… -D… -classpath
/path/to/local/distribution/lib/gradle-launcher
.jar org.gradle.launcher.GradleMain SomeTask
```

…

After the **`exec`** command, the shell process turns into a **`java`** process, i.e. JVM process

… run the Java code in **`org.gradle.launcher.GradleMain`**

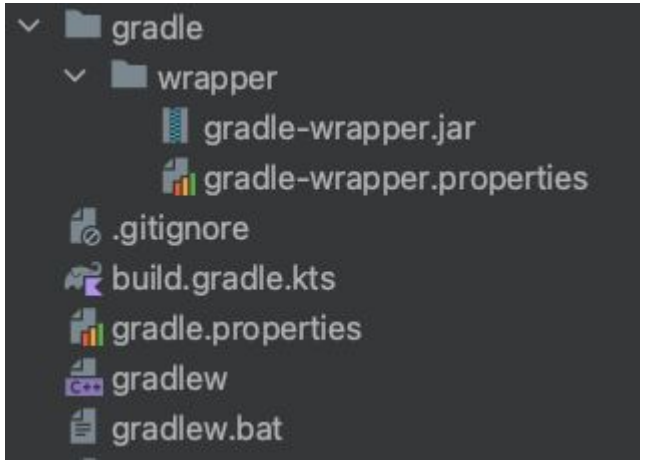




https://docs.gradle.org/current/userguide/gradle_daemon.html#compatibility
https://blog.gradle.org/how-gradle-works-1

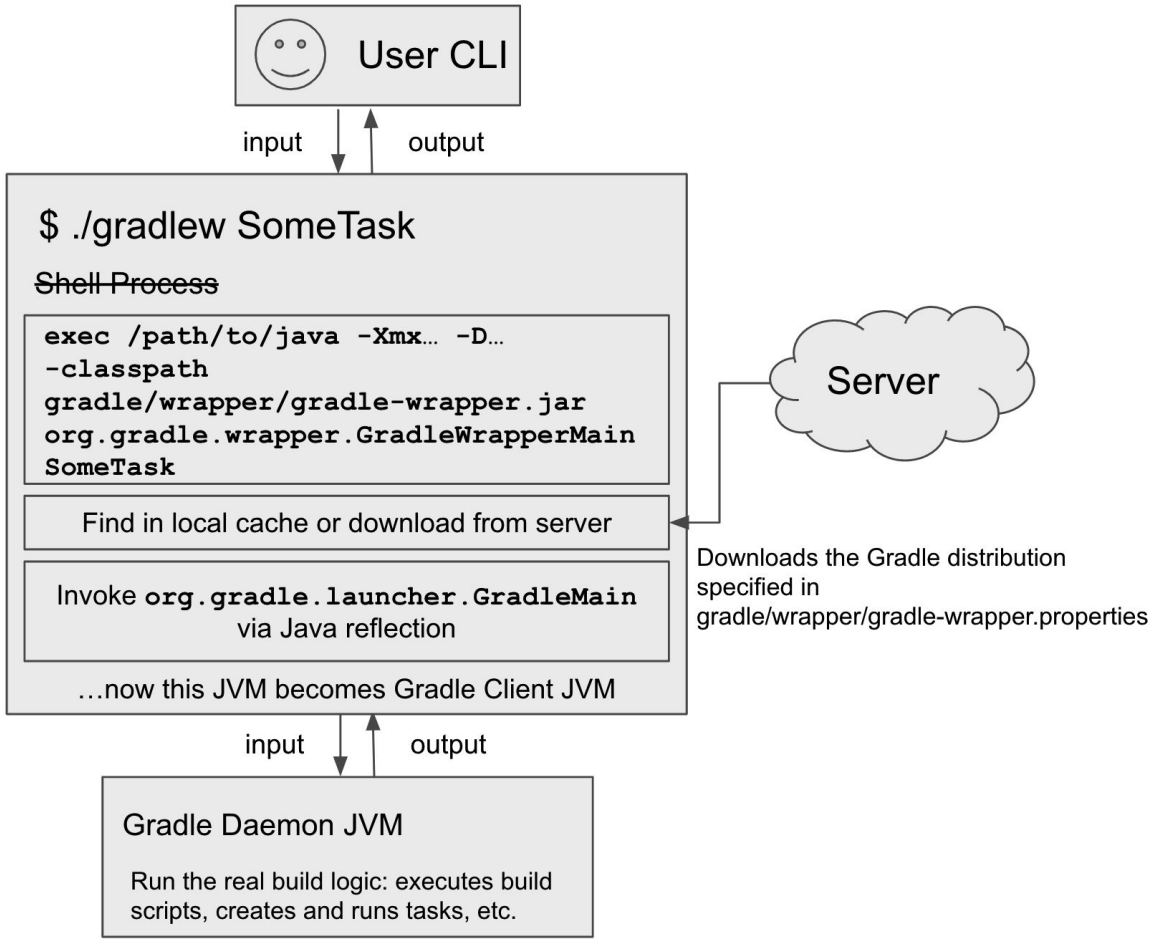
./gradlew someTask

Запуск через Gradle, который используется в проекте с помощью Gradle Wrapper
Добавляется в проект автоматически либо командой gradle wrapper, который добавит:
1. Скрипты для запуска билда в зависимости от OS
2. Properties для того, чтобы задекларировать версию gradle и ссылку для скачивания
3. JARник для скачивания Gradle



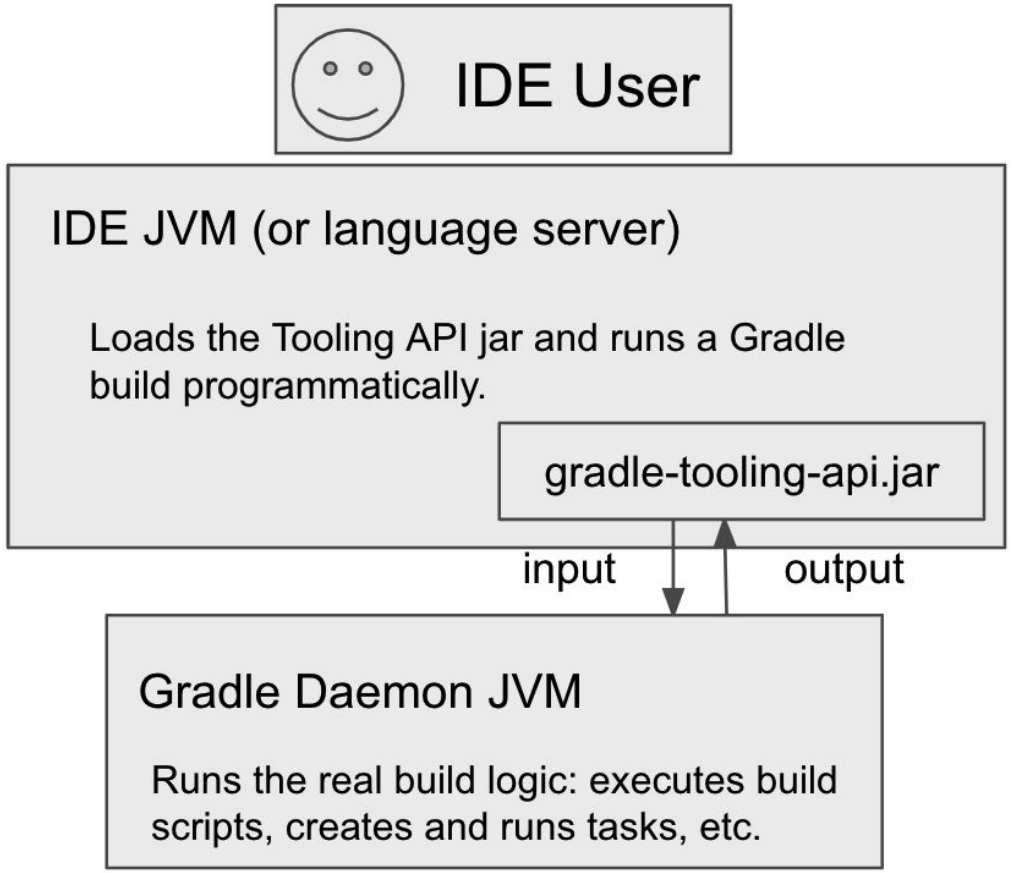

```
#Sat Mar 04 08:16:20 MSK 2023
distributionBase=GRADLE_USER_HOME
distributionPath=wrapper/dists
distributionUrl=https\://services.gradle.org/distributions/gradle-7.5-bin.zip
zipStoreBase=GRADLE_USER_HOME
zipStorePath=wrapper/dists
```

./gradlew someTask



https://blog.gradle.org/how-gradle-works-1

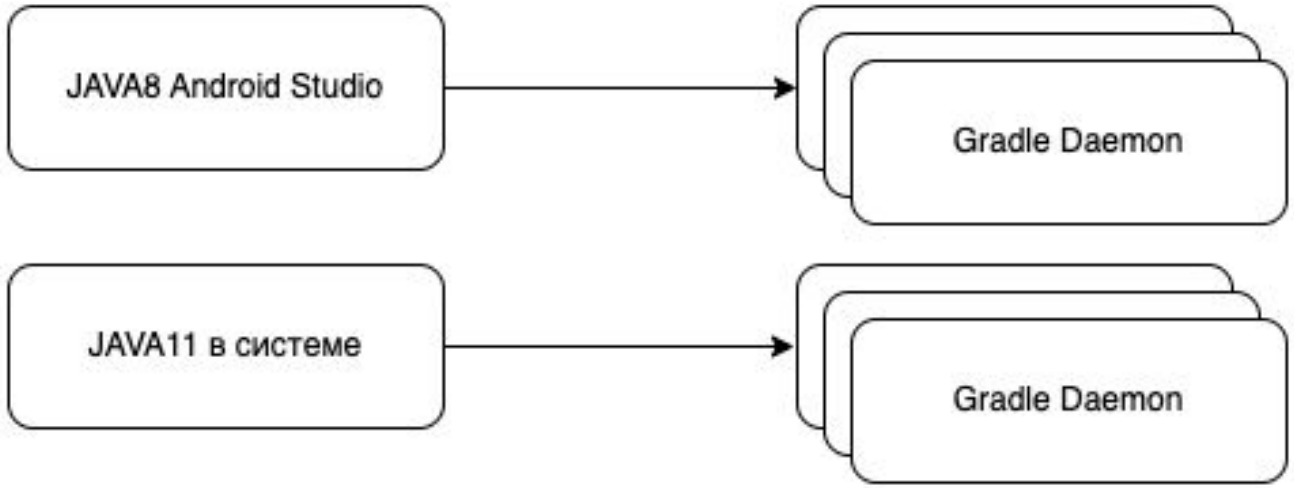
Кнопка в Android Studio



https://blog.gradle.org/how-gradle-works-1
https://docs.gradle.org/current/userguide/third_party_integration.html#embedding

Что может пойти не так?

- Использование разных версии java в командной строке и в Android Studio приводит к тому, что появляется больше демонов, чем нужно
- Необходимо прибивать ненужных демонов:
  - ./gradlew --stop для одной Gradle
  - pkill -f '.*GradleDaemon.*' для всех
- Используйте Gradle Doctor для определения других проблем



https://runningcode.github.io/gradle-doctor/java-home/

Android Gradle Plugin (AGP)

AGP

- Плагин для сборки Android приложений от Google
- Содержит в себе разные плагины по типу application/library
- Сильно зависит от версии Android Studio, Build Tools, JDK, Gradle

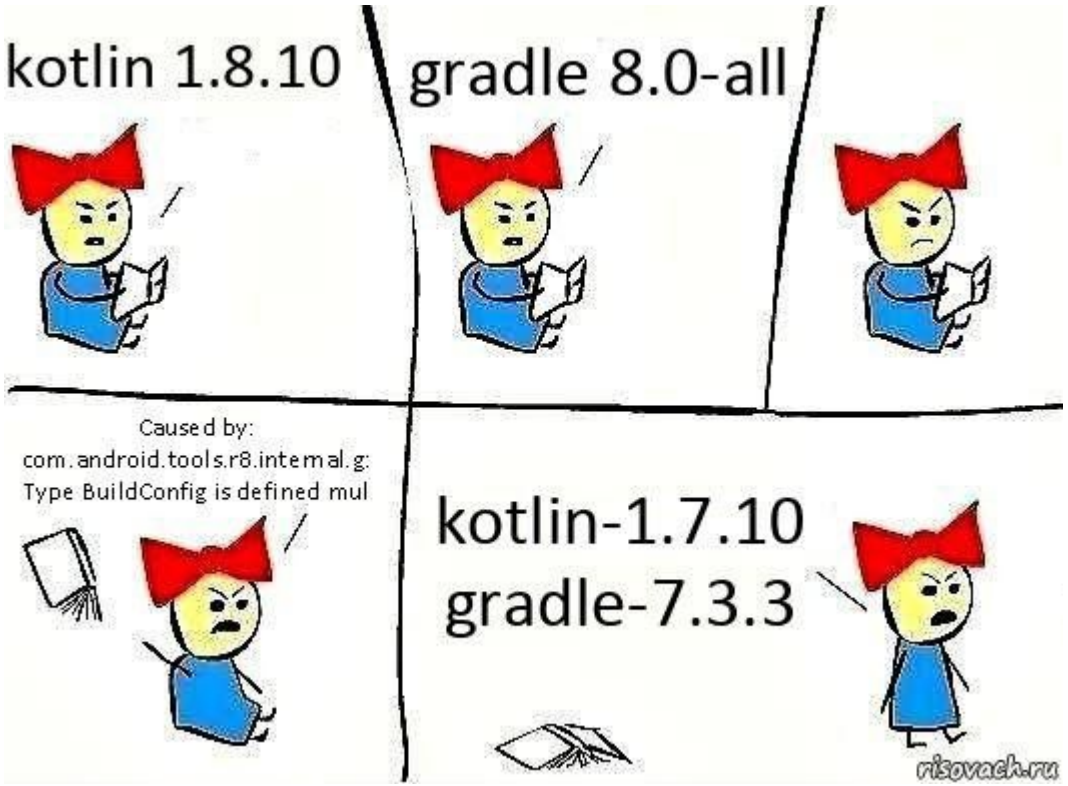
Android Studio version	Required AGP version
Jellyfish | 2023.3.1	3.2-8.4
Iguana | 2023.2.1	3.2-8.3
Hedgehog | 2023.1.1	3.2-8.2
Giraffe | 2022.3.1	3.2-8.1
Flamingo | 2022.2.1	3.2-8.0
Electric Eel | 2022.1.1	3.2-7.4

Plugin version	Minimum required Gradle version
8.4 (alpha)	8.6-rc-1
8.3	8.4
8.2	8.2
8.1	8.0
8.0	8.0
7.4	7.5

	Minimum version	Default version
Gradle	8.4	8.4
SDK Build Tools	34.0.0	34.0.0
NDK	N/A	25.1.8937393
JDK	17	17

https://developer.android.com/build/releases/gradle-plugin#updating-gradle

Болезненные обновления



Что можно сконфигурировать с помощью AGP?

- compileSdkVersion
- minSdkVersion
- targetSdkVersion
- Build Type
- Product Flavor
- Build Variant
- Build Config
- Sources Set
- Manifest Transformation
- Изменение исходников в процессе сборки
- Подпись приложения
- Обфускация и оптимизация кода

Sdk Versions

- **compileSdkVersion**
  - Указывается версия sdk, которая будет использоваться при компиляции приложения
- **minSdkVersion**
  - Минимальная поддерживаемая вашим устройством версия.
  - Чем оно ниже compileSdkVersion, тем больше надо будет писать if-else и писать обратно совместимый код, иногда лишая себя нового функционала в Android
- **targetSdkVersion**
  - Определяет версию, для которой тестировалось приложение и оно готово ко всем изменениям в новой версии
  - Если приложение имеет targetSdkVersion меньше, чем в системе, то Android включит режим обратной совместимости и некоторые изменения не будут применены для приложения.
  - Не все изменения будут откатаны. Например, изменения Android Runtime Permissions с временной выдачей разрешений или их откатом будет применен независимо от версии.
  - Google Play постоянно ставит дедлайны на повышение версии target sdk

https://youtu.be/JEAUTGYmz2s
https://apptractor.ru/develop/compilesdkversion-i-targetsdkversion-v-chem-raznitsa.html

Build Type

Определяет конфигурацию проекта для разных этапов разработки.
buildType может определять

- возможность отладки
- ключи для подписи
- как обфусцировать код
- sourceSets
- build config
- таски для сборки проекта
- matchingFallbacks
- конфигурации для зависимостей
- изменять значения в manifest (Manifest Placeholder)

По умолчанию есть два разных типа: debug и release, но должен быть как минимум один

```
debugImplementation("com.facebook.flipper:flipper:0.183.0")
releaseImplementation("com.facebook.flipper:flipper-noop:0.183.0")
```

```
buildTypes { this: NamedDomainObjectContainer<ApplicationBuildType>
    named( name: "release") { this: ApplicationBuildType
        isMinifyEnabled = true
        setProguardFiles(
            listOf(
                getDefaultProguardFile( name: "proguard-android-optimize.txt"),
                "proguard-rules.pro"
            )
        )
        addManifestPlaceholders(
            mapOf(
                "networkSecurityConfig" to "@xml/release_network_security_config",
            )
        )
    }
    named( name: "debug").configure { this: ApplicationBuildType
        applicationIdSuffix = ".debug"
        isMinifyEnabled = false
        addManifestPlaceholders(
            mapOf(
                "networkSecurityConfig" to "@xml/debug_network_security_config",
            )
        )
    }

    // если в модуле не нужны некоторые buildType. Например beta, то можно одно приравнять к другому
    maybeCreate( name: "beta").apply { this: ApplicationBuildType
        matchingFallbacks.add("release")
    }
    maybeCreate( name: "qa").apply { this: ApplicationBuildType
        buildConfigField( type: "String",
            name: "BUILD_TYPE_DESCRIPTION",
            value: "TestingBuildType")
        buildConfigField( type: "int",
            name: "BUILD_TYPE_VERSION",
            value: "99")
    }
}
```

Product Flavor

Разные версии для конечного пользователя через создание специальных dimension

flavor может определять

- практически то же, что и **buildType**

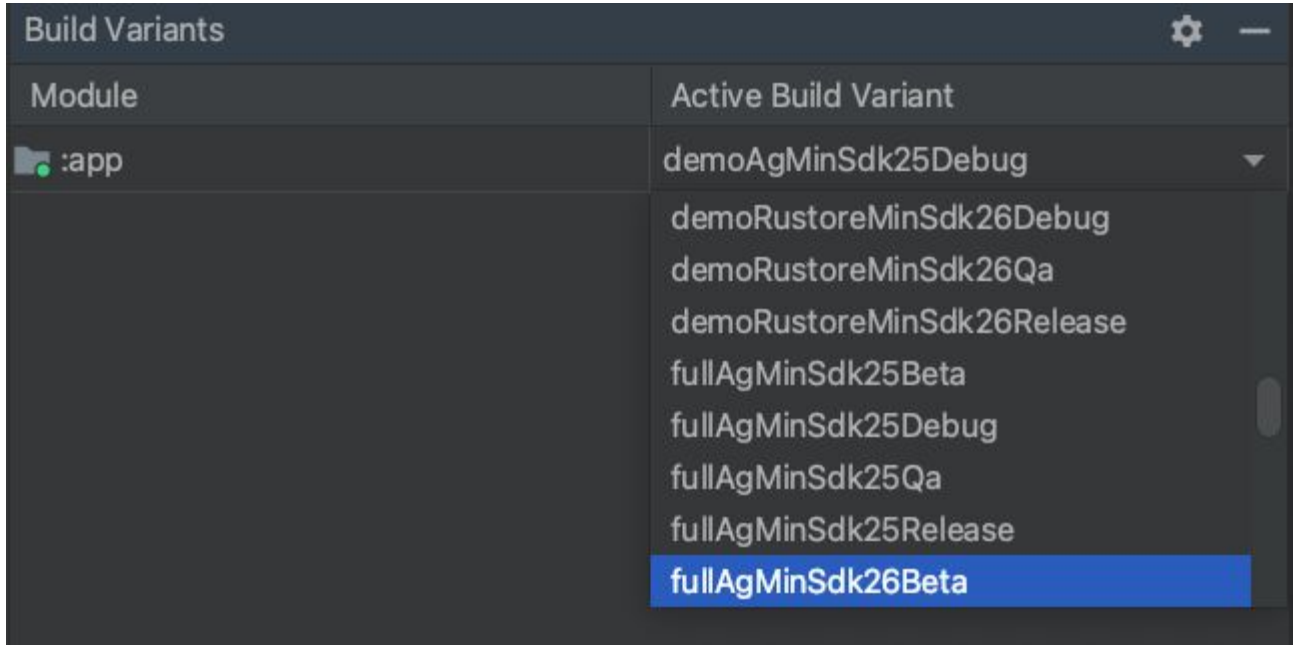
По умолчанию их нет

```
flavorDimensions += listOf("version", "market", "minSdk")
productFlavors { this: NamedDomainObjectContainer<ApplicationProductFlavor>
    create( name: "demo") { this: ApplicationProductFlavor
        dimension = "version"
        applicationIdSuffix = ".demo"
        versionNameSuffix = "-demo"

        addManifestPlaceholders(
            mapOf(
                "networkSecurityConfig" to "@xml/debug_network_security_config",
            )
        )
    }
    create( name: "full") { this: ApplicationProductFlavor
        dimension = "version"
        applicationIdSuffix = ".full"
        versionNameSuffix = "-full"
    }
    create( name: "gp") { this: ApplicationProductFlavor
        dimension = "market"
        applicationIdSuffix = ".gp"
        versionNameSuffix = "-gp"
    }
    create( name: "ag") { this: ApplicationProductFlavor
        dimension = "market"
        applicationIdSuffix = ".ag"
        versionNameSuffix = "-ag"
    }
    create( name: "rustore") { this: ApplicationProductFlavor
        dimension = "market"
        applicationIdSuffix = ".rustore"
        versionNameSuffix = "-rustore"
    }

    create( name: "minSdk25") { this: ApplicationProductFlavor
        dimension = "minSdk"
        minSdk = 25
    }
    create( name: "minSdk26") { this: ApplicationProductFlavor
        dimension = "minSdk"
        minSdk = 26
    }
}
```

Build Type + Product Flavor = Build Variant



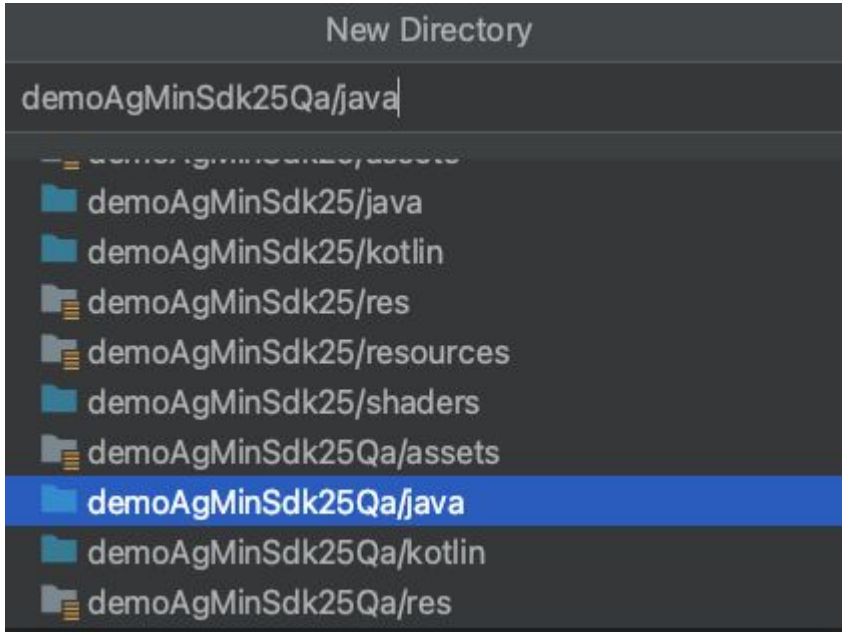
Disable Build Variant

- Минимизируйте количество **flavor** и **buildVariant**
- Отключайте ненужные пересечения для быстрой синхронизации проекта

```
android { this: BaseAppModuleExtension
    androidComponents.beforeVariants { it
        if (it.name == "demoRelease") {
            it.enable = false
        }
    }
}
```

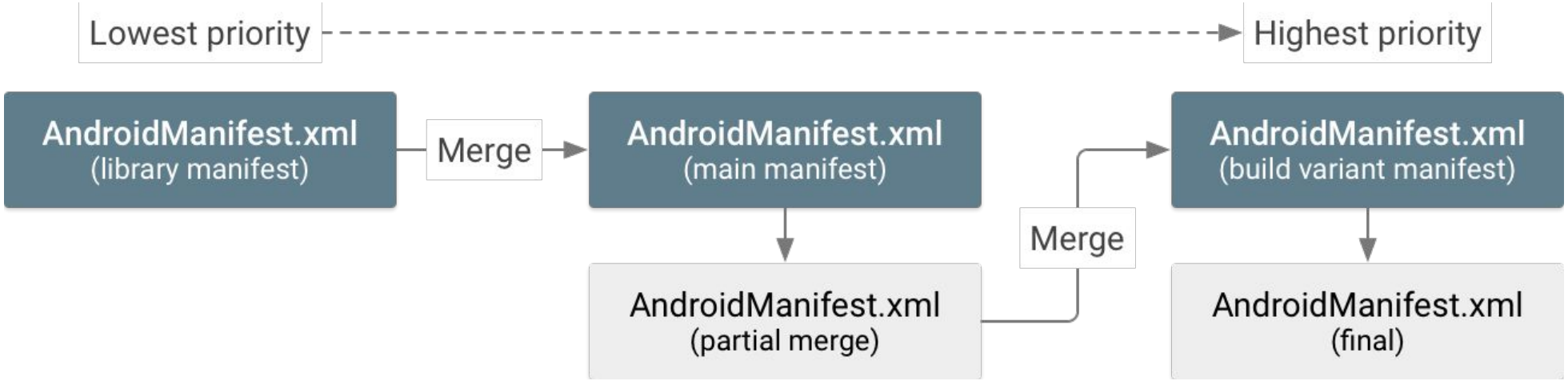
SourceSet

- Папки с кодом и ресурсами для **buildVariant**
- Можно добавлять свой
- sourceSets одного **buildType** не должно быть одинаковых классов, иначе сборка упадет



Manifest Merging

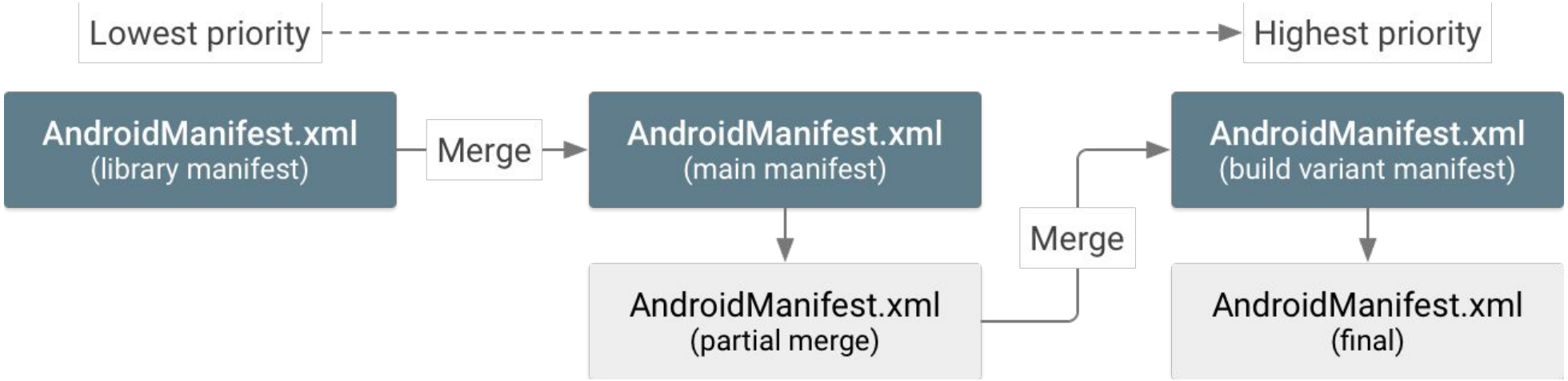
- Все манифесты собираются в один единый
- Мерж происходит по приоритетам
- Есть специальные правила **tools:node**, которые позволяют некоторое поведение починить либо переопределить



https://developer.android.com/studio/build/manage-manifests

Приоритеты Manifest Merging

- Манифест нашего buildType
  - Манифест из sourceSet для buildVariant
  - Манифест из sourceSet для buildType
  - Манифест из sourceSet для productFlavor
    - Если их несколько, то по порядку их объявления в flavorDimensions
- Манифест Android Application Module
- Манифесты подключаемых библиотек



https://developer.android.com/studio/build/manage-manifests

Результат работы AGP

- Manifests -> Merged манифест (один большой)
- Код -> .classes -> dex
- Ресурсы -> скомпилированные ресурсы и resources.arsc таблица идентификаторов

Многомодульность

А надо ли?

- Что она даст?
- Превратит кучу плохо кода в много куч плохого кода?
- Навредит или наоборот поможет?
- Разработчики мешают друг другу?
- Много конфликтов?
- Долгая сборка?
- Покупка более мощного устройства или аренда сервера не решит проблему?

План

- Определить преимущества и недостатки
- Определить структуру компании, команды
- Выделить типы модулей
- Определить их ограничения
- Определить ответственных за модули
- Настройка удобной конфигурации

Преимущества

- Включает преимущества системы сборки
- Горизонтальная расширяемость
- Быстрая инкрементальная сборка
- Compilation Avoidance
- Распределение ответственности за модули
- Уменьшение связанности компонентов

Проблемы

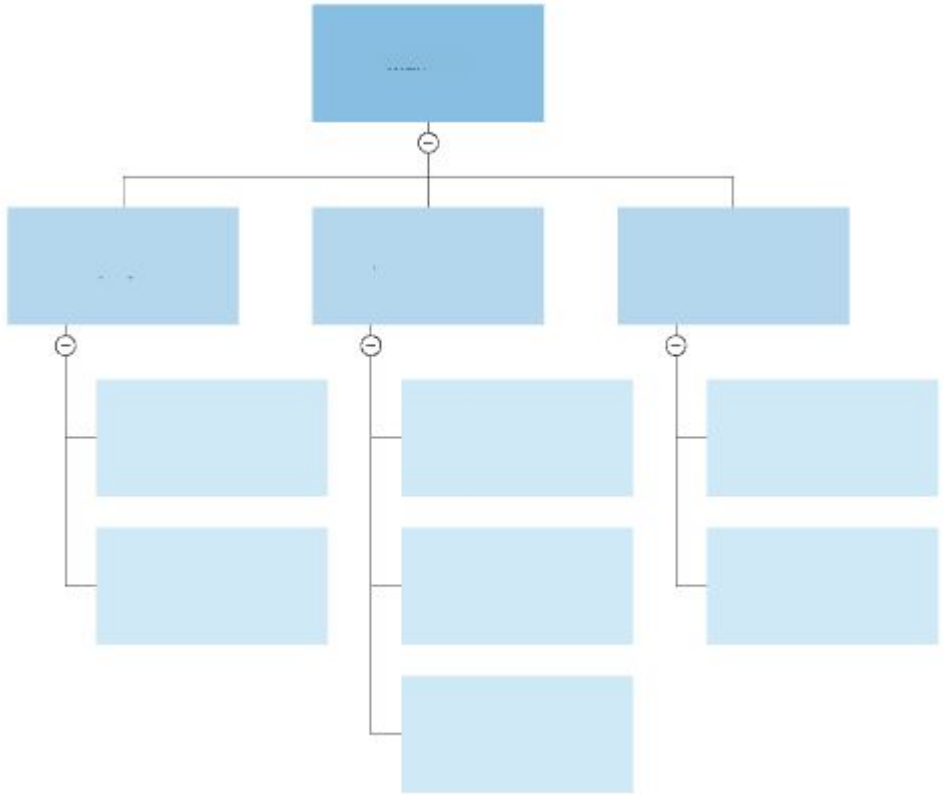
- Сложно ориентироваться и найти нужный код.
- Сложность управления зависимостями (тут помогут способы хранения версии зависимостей из части про Gradle)
- Копипаста конфигурационных файлов build.gradle.kts и settings.gradle.kts
- Сложно понимать конфигурационные файлы
- Заброшенные модули без привязки к кому-то
- Структура модулей не позволяет расширяться
- Могут появиться циклические зависимости
- Долгая сборка из-за неудачных связей
- Сильная связанность из-за отсутствия достаточных ограничений на зависимости (Фича поверх фичи)
- Много мерж конфликтов

https://habr.com/ru/company/cian/blog/662766/

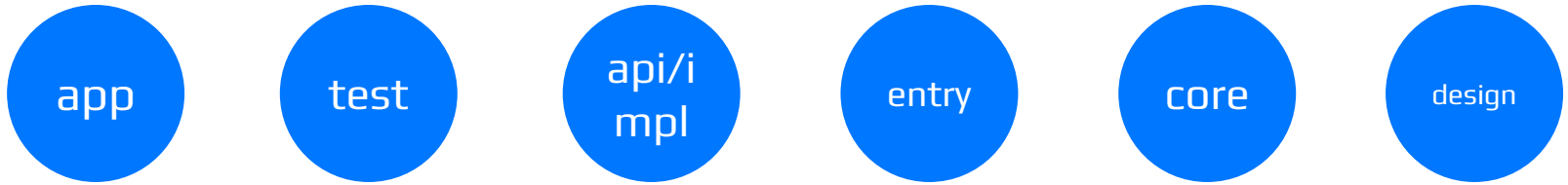
Структура

Проанализируйте:

- компанию и ее архитектуру
- команду
- реализуемый функционал приложения
- рост проекта



Типы модулей



app

- модуль с плагином android application
- по нему собирается итоговый apk
- может быть несколько, если несколько приложений в одном проекте
- подключает к себе все остальные модули

api/impl

- два модуля - с интерфейсами и их реализациями
- api модуль содержит интерфейсы, которые доступны другим клиентам, поэтому к его наполнению надо подходить осторожно
- impl содержит реализации и этот модуль никуда не должен подключаться кроме entry и app
- при активной разработке, клиентские модули не пересобираются, потому что меняется только impl, а не api
- такой подход в большинстве избавляет от циклических зависимостей
- api к api стоит подключать аккуратно. 10 раз подумав

core

- содержит что-то базовое, что нужно всем. работа с сетью, di, хранилищем, тогглы, утилиты и прочее
- может быть разделение на api и impl
- может быть оберткой над разными библиотеками и предоставлять один интерфейс взаимодействия
- можно подключать ко всем модулям, но если есть api и impl часть, то действуют правила api/impl модулей

test

- мог быть и в core, но почему бы не выделить
- содержит утилиты для тестирования кода или тесты по типу benchmark
- должен подключаться через testImplementation или androidTestImplementation
- не должен попасть в итоговый apk

design

- самый простой для начала модуль. можно сразу выносить общие ui компоненты
- по своей сути так же core, но вынесен в отдельный тип по своей значимости
- подключается к модулям с ui

entry

- опциональный модуль
- основная задача разгрузить app модуль, чтобы его build.gradle занимался только настройкой проекта
- например, тут можно описывать di связи, навигацию и прочее
- может быть мостом между одной фичей и множеством других
- по сути это шина, но которая не должна содержать какой-то своей логики

Ограничения на зависимости

По типам модулей.

- impl -> entry/app
- api -> impl/entry (к api осторожно)
- core -> ко всем
- design-> impl/core/app (там, где есть ui)
- entry -> app

По любым другим параметрам.
Например, запрет на Compose для определенных команд.

Проверки можно реализовать через Gradle Plugin

Ответственность за модули

- Группой модулей может владеть команда
- но у каждого модуля должен быть как минимум один владелец

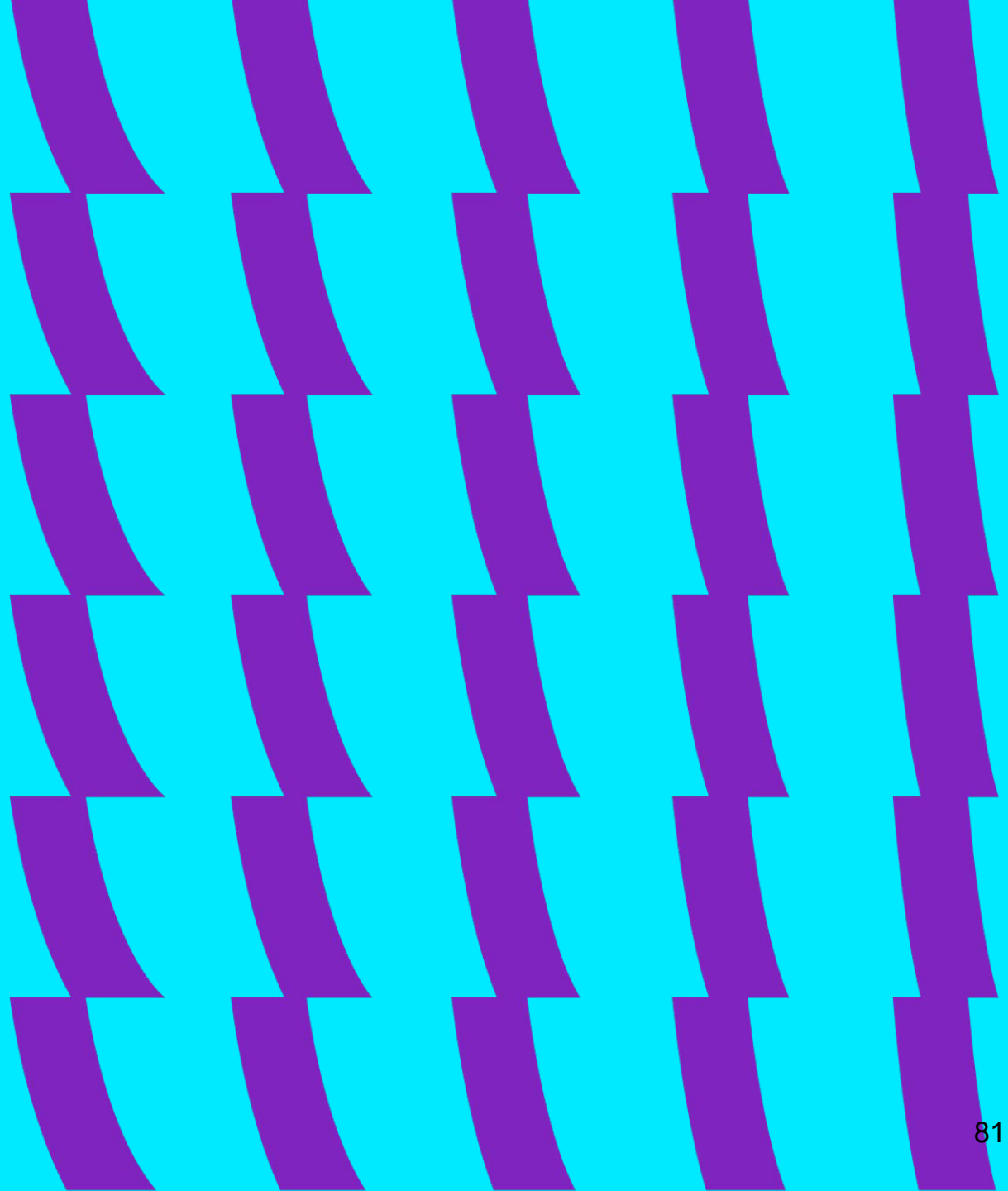
Правила конфигурации модулей

- должно быть максимальное упрощение **build.gradle.kts** для модулей
- правила должны не допускать излишней копипасты
- использовать больше implementation и редких случаях api конфигурацию
- применить один из способов организации хранения зависимостей

хороший пример плагина:
https://github.com/formatools/forma

```
// Single method, type-safe creation of your target
// Plugins applied automatically
// Project configuration shared between targets
androidLibrary(
    // Mandatory, visible from build configuration
    packageName = "com.stepango.example",
    // External dependencies declaration, one universal syntax
    dependencies = deps(
        google.material,
        androidx.appcompat,
    ) + deps(
        // Internal project dependencies, declared separately from externals
        project(":demo-library")
    ),
    // Test dependencies declaration
    testDependencies = deps(
        test.junit
    ),
    // Android test dependencies declaration
    androidTestDependencies = deps(
        test.espresso
    )
)
```

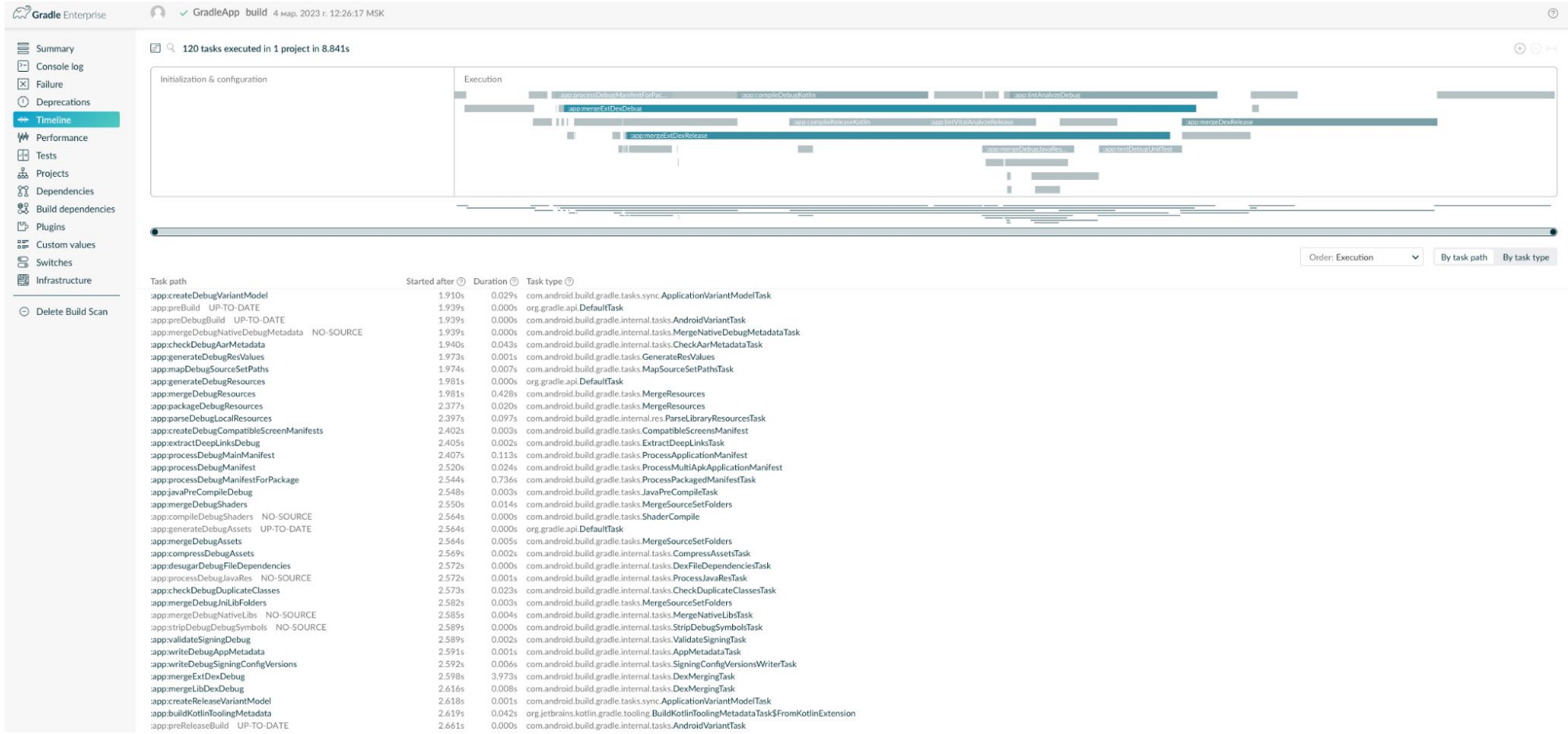
Профайлинг



Gradle Build Scan

root build.gradle.kts

```
if (hasProperty("buildScan")) {
    buildScan {
        termsOfServiceUrl =
"https://gradle.com/terms-of-service"
        termsOfServiceAgree = "yes"
    }
}
```



gradle-profiler

Сценарий сравнения сборки с библиотеками для работы с БД

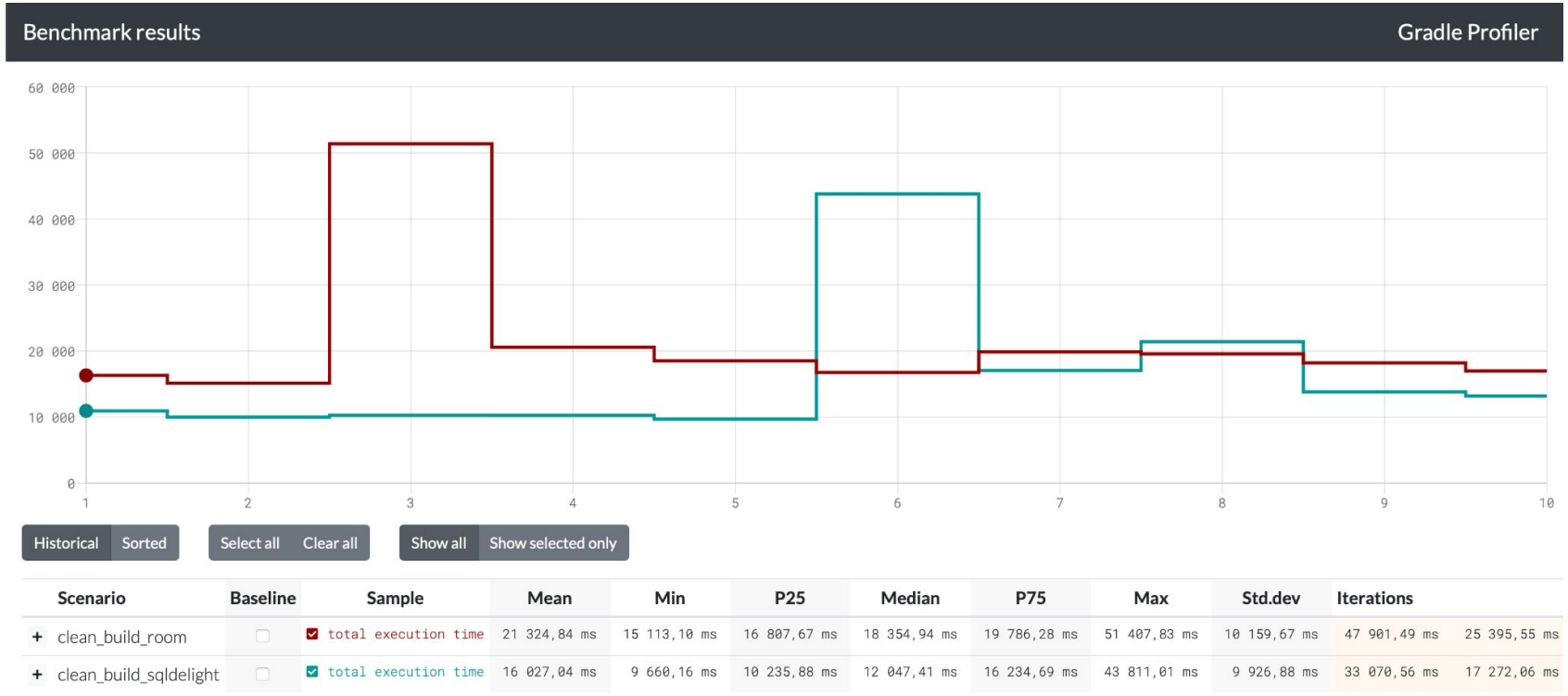
```
clean_build_sqldelight {
    tasks = [":core:database:assemble"]
    cleanup-tasks = ["clean"]
    gradle-args = ["-Dorg.gradle.caching=false  --rerun-tasks --no-build-cache --no-configuration-cache"]
    git-checkout = {
        build = "sqldelight"
    }
}

clean_build_room {
    tasks = [":core:database:assemble"]
    cleanup-tasks = ["clean"]
    gradle-args = ["-Dorg.gradle.caching=false  --rerun-tasks --no-build-cache --no-configuration-cache"]
    git-checkout = {
        build = "room"
    }
}
```

Утилита: https://github.com/gradle/gradle-profiler
Репозиторий со сценарием:https://github.com/0xera/nowinandroid-sqldelight

gradle-profiler

Запуск: gradle-profiler --bechmark --scenario compare-file.scenarios
Результат на 10 прогонов:



Scenario	Baseline	Sample	Mean	Min	P25	Median	P75	Max	Std.dev	Iterations	
+ clean_build_room		total execution time	21 324,84 ms	15 113,10 ms	16 807,67 ms	18 354,94 ms	19 786,28 ms	51 407,83 ms	10 159,67 ms	47 901,49 ms	25 395,55 ms
+ clean_build_sqldelight		total execution time	16 027,04 ms	9 660,16 ms	10 235,88 ms	12 047,41 ms	16 234,69 ms	43 811,01 ms	9 926,88 ms	33 070,56 ms	17 272,06 ms

Утилита: https://github.com/gradle/gradle-profiler
Репозиторий со сценарием:https://github.com/0xera/nowinandroid-sqldelight

Pure Java/Kotlin > Android

Стоит отдавать предпочтение java/kotlin модулям без AGP, так как AGP иногда выполняет лишнюю работу для конкретного модуля.

Если нужны Android классы просто, чтобы собраться, то можно подключить android зависимость как compile only.
Это Java зависимость со stub классами.

```
compileOnly("com.google.android:android:4.1.1.4")
```

Не дошли

- как устроены конфигурации в Gradle
- как работает compilation avoidance
- что такое ABI
- и тд

Спасибо за внимание

Полезные ссылки

https://threadreaderapp.com/thread/1466134453940629515.html
https://www.bruceeckel.com/2021/01/02/the-problem-with-gradle/
https://melix.github.io/blog/2021/01/the-problem-with-gradle.html
https://blog.gradle.org/gradle-vs-bazel-jvm
https://developer.squareup.com/blog/stampeding-elephants/
https://docs.gradle.org/current/userguide/plugins.html#sec:types_of_plugins
https://docs.gradle.org/8.0/release-notes.html
https://docs.gradle.org/8.0/userguide/plugins.html#sec:constrained_syntax
https://docs.gradle.org/current/userguide/organizing_gradle_projects.html#sec:build_sources
https://funkymuse.dev/posts/toml-gradle/
https://docs.gradle.org/current/userguide/platforms.html
https://docs.gradle.org/current/userguide/gradle_daemon.html#compatibility
https://docs.gradle.org/current/userguide/third_party_integration.html#embedding
https://apptractor.ru/develop/compilesdkversion-i-targetsdkversion-v-chem-raznitsa.html
https://developer.android.com/studio/build/manage-manifests
https://github.com/formatools/forma
https://github.com/gradle/gradle-profiler
https://youtu.be/JEAUTGYmz2s
https://habr.com/ru/company/cian/blog/715640/
https://habr.com/ru/company/cian/blog/667776/
https://habr.com/ru/company/cian/blog/662766/
https://habr.com/ru/company/cian/blog/670468/
https://habr.com/ru/company/cian/blog/672862/
https://youtu.be/Amjn9gVvPHo
https://www.youtube.com/watch?v=Amjn9gVvPHo
https://www.youtube.com/watch?v=yDj0n0g5dXY
https://www.youtube.com/watch?v=hpT0VutO5xk
https://www.youtube.com/watch?v=8C4tJMxjb_4
https://www.youtube.com/watch?v=WOBok2u-SL8
https://youtu.be/lzaFHmac5_k
https://www.youtube.com/watch?v=o0M4f5djJTQ
https://www.youtube.com/watch?v=p7-AffMucBw
https://www.youtube.com/watch?v=2SWgl-OdxDY
https://developer.android.com/studio/build/configure-app-module