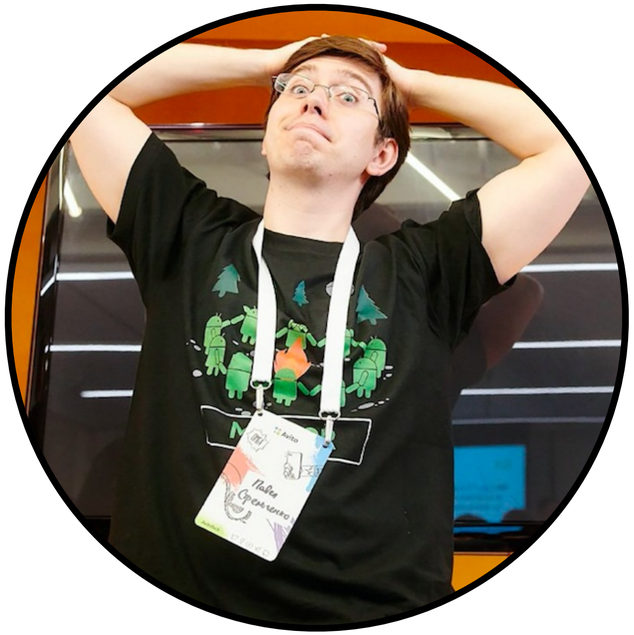
Укрощение feature-флагов

Павел Стрельченко

Павел Стрельченко



Android-разработчик в HeadHunter

Пишу плагины для Android Studio

Помогаю коллегам работать продуктивно

О чем сегодня будем говорить?

1. Проблемы feature-флагов
2. Уход от merge-конфликтов
3. Способы сбора флагов по всей кодовой базе

Ссылка на слайды

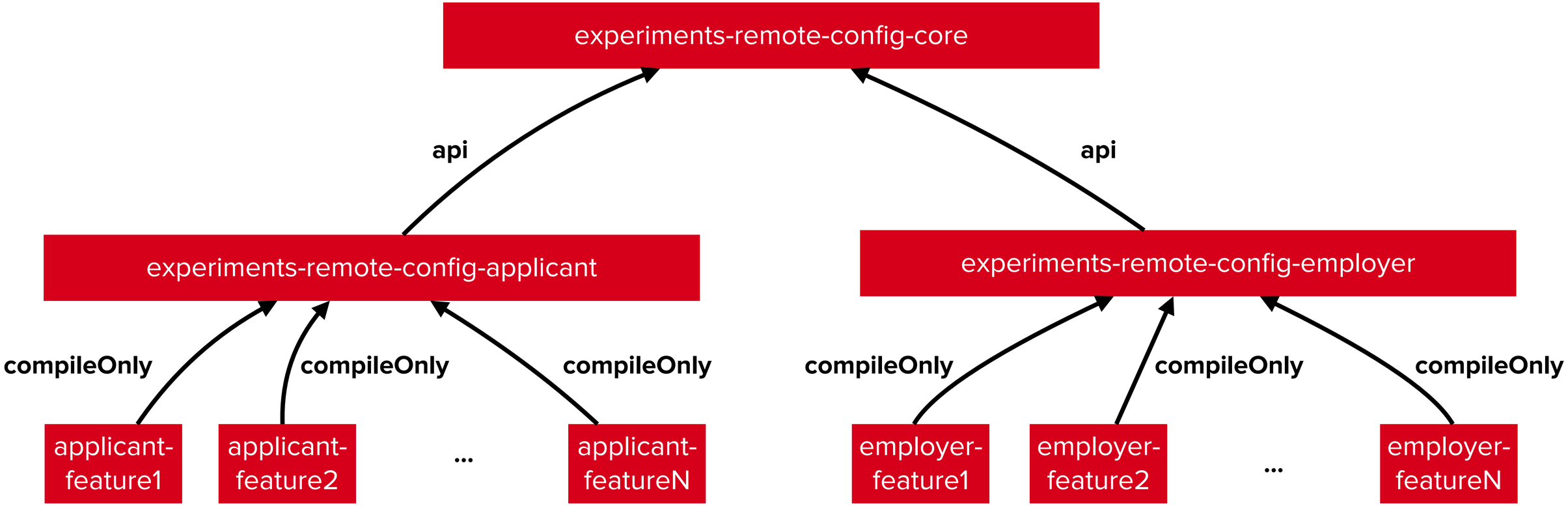

Проблемы
feature-флагов

Немного специфики hh.ru

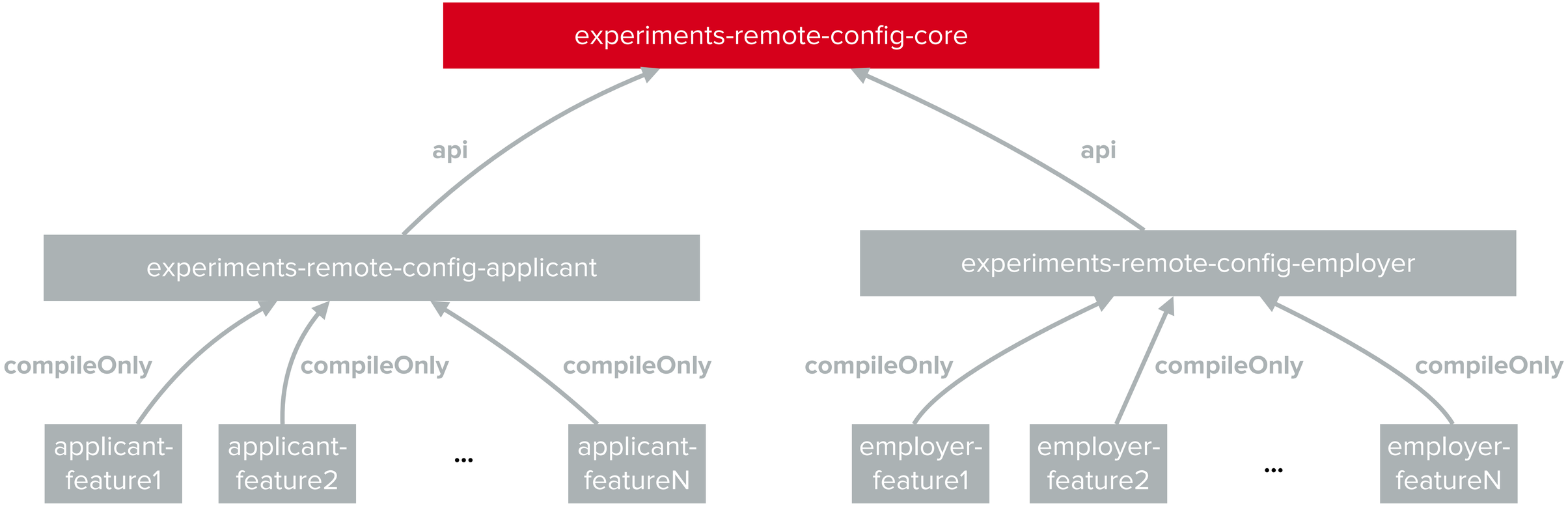
1. Мы называем feature-флаги «экспериментами»

2. Год назад у нас было около 10 разработчиков, активно пишущих код

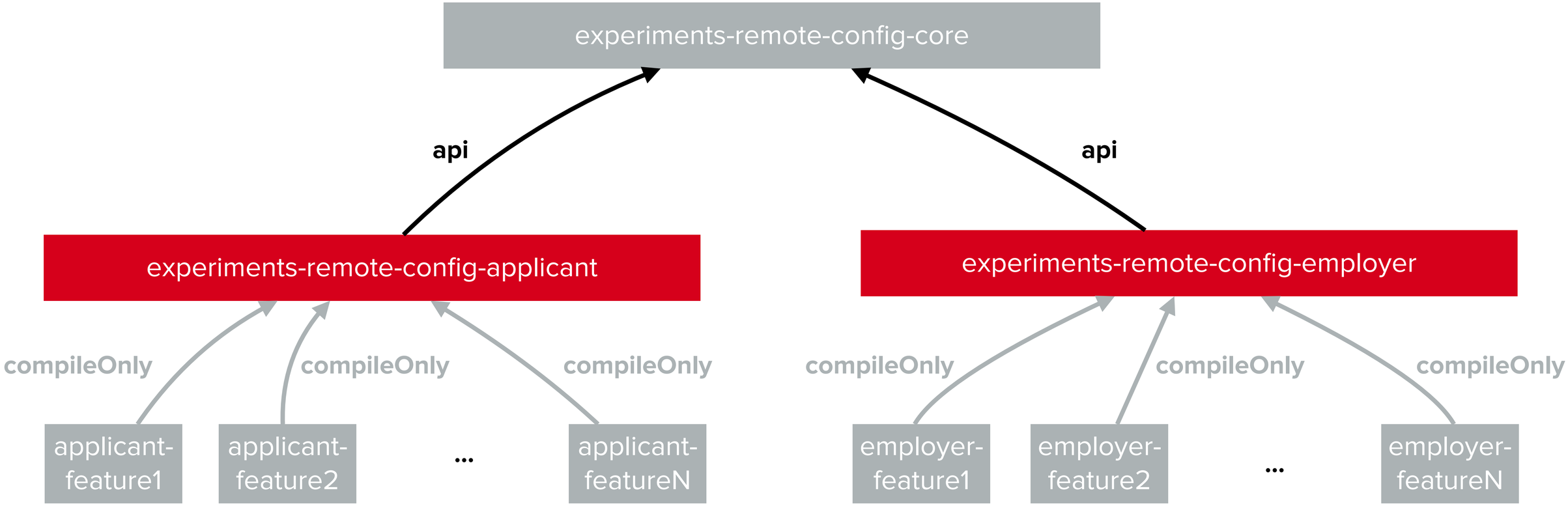
Первая реализация «конфига»



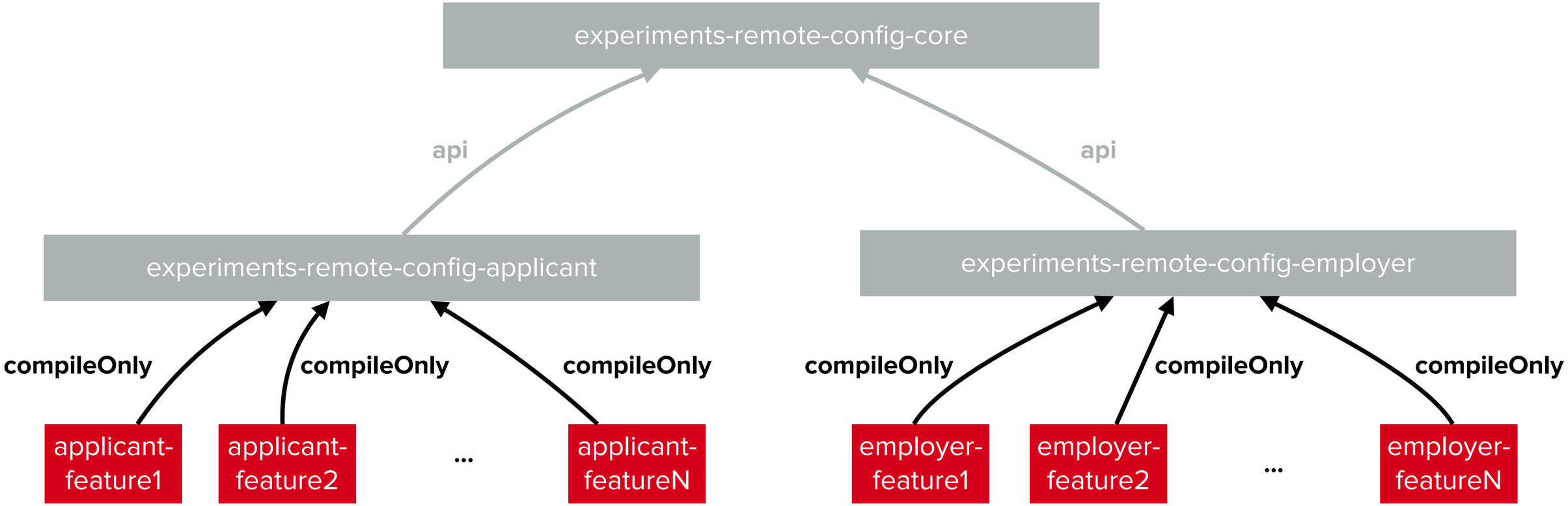
Базовая логика экспериментов



Эксперименты для приложений



Использование экспериментов



Три основные проблемы

1. **Постоянные merge-конфликты в общем наборе экспериментов**
2. Пересборка почти всего приложения при добавлении эксперимента
3. Ради проверки эксперимента писали очень много кода

Merge-конфликты в enum-е

```
enum class ApplicantExperiments(
        val key: String,
        val description: String = String.EMPTY
) {
    EXPERIMENT_APPS_CLUSTERS_ON_JOBSEARCH( key: "apps clusters on jobsearch"),
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_CLUSTERS( key: "android_quick_filters_and_clusters"),
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_ADVANCED_SEARCH( key: "android_quick_filters_and_advanced_search"),
    EXPERIMENT_ANDROID_FILTERS_AND_ADVANCED_SEARCH( key: "android_filters_and_advanced_search"),
    EXPERIMENT_ANDROID_WANT_TO_WORK_AND_SUBSCRIBE( key: "android_want_to_work_and_subscribe"),
    EXPERIMENT_ANDROID_ANDROID_WANT_TO_WORK_ONLY( key: "android_want_to_work_only"),
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_COMPANY( key: "android_response_btn_after_company"),
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_ADDRESS( key: "android_response_btn_after_address"),
    EXPERIMENT_ANDROID_MORE_ADS_DIRECT( key: "android_more_ads_direct"),
    EXPERIMENT_ANDROID_RENAME_RESPONSE_TO_WANT_TO_WORK( key: "android_rename_response_to_wtw"),
    EXPERIMENT_ANDROID_AUTO_OPEN_EDIT_WHEN_NO_RESUME( key: "auto_open_edit_when_no_resume"),
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTLINE( key: "gh_an_vacancies_response_btn_B"),
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTLINE_C( key: "gh_an_vacancies_response_btn_C"),
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTLINE_D( key: "gh_an_vacancies_response_btn_D"),
    EXPERIMENT_ANDROID_PROFILE_NATIVE_COMPLETION( key: "app_android_native_completion"),
    EXPERIMENT_GH_ANDROID_MAIN_REMOVE_TITLE( key: "gh_android_main_remove_title"),
    EXPERIMENT_GH_ANDROID_MAIN_INCREASE_VACANCY_COUNT( key: "gh_android_main_increase_vacancy"),
    EXPERIMENT_GH_ANDROID_MAIN_REMINDER( key: "gh_android_main_reminder"),
    EXPERIMENT_ANDROID_SHOW_RESUME_STATISTICS( key: "android_show_resume_statistics"),
    EXPERIMENT_ANDROID_SHOW_RESUME_STATISTICS_WITHOUT_EMOJI( key: "andr_resume_stat_without_emoji"),
    EXPERIMENT_GH_ANDROID_FAVORITES_TAB_RENAME( key: "gh_android_favorites_tab_rename"),
    EXPERIMENT_GH_ANDROID_SIMILAR_VACANCIES_LARGE( key: "gh_andr_similar_vacancies_large"),
    EXPERIMENT_GH_ANDROID_SIMILAR_VACANCIES_LARGE_AND_INCREASE_COUNT( key: "gh_andr_similar_vacancies_10"),
    EXPERIMENT_ANDROID_PROFILE_COMPLETION_REPLACE_CREATION( key: "app_droid_complet_replace_create"),
    EXPERIMENT_GH_AN_MAIN_ARTICLES( key: "gh_an_main_articles"),
    EXPERIMENT_GH_AN_MAIN_STORIES( key: "gh_an_main_stories"),
    EXPERIMENT_GH_AN_MAIN_STORIES_JUST_NEWS( key: "gh_an_main_stories_just_news"),
    EXPERIMENT_APPS_REGISTRATION_WITH_SOCIAL( key: "apps_reg_with_social"),
    EXPERIMENT_APPS_REGISTRATION_WITH_SOCIAL_COLLAPSED( key: "apps_reg_with_social_collapsed"),
    EXPERIMENT_GH_ANDROID_REMOTE_BLOCK_IN_RECOMMEND_TAB( key: "gh_an_main_remote"),
    EXPERIMENT_GH_ANDROID_SUGGEST_SEARCH_HISTORY( key: "gh_an_suggest_search_history"),
```

Merge-конфликты в enum-е

```
enum class ApplicantExperiments(
        val key: String,
        val description: String = String.EMPTY
) {
    EXPERIMENT_APPS_COMPLETION_ON_RESPONSE("apps compl
    EXPERIMENT_APPS_CLUSTERS_ON_JOBSEARCH("apps cluste
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_CLUSTERS("and
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_ADVANCED_SEAR
    EXPERIMENT_ANDROID_FILTERS_AND_ADVANCED_SEARCH("an
    EXPERIMENT_ANDROID_WANT_TO_WORK_AND_SUBSCRIBE("and
    EXPERIMENT_ANDROID_ANDROID_WANT_TO_WORK_ONLY("andr
    EXPERIMENT_APPS_REGION_ON_MAIN_SCREEN("apps_region
    EXPERIMENT_APPS_NATIVE_OK_LOGIN("apps_native_ok_lo
    EXPERIMENT_APPS_NATIVE_VK_LOGIN("apps_native_vk_lo
    EXPERIMENT_APPS_CHANGE_POSITION_FROM_TITLE("apps_c
    EXPERIMENT_NEW_RESPONSE_SCREEN("app_new_response_s
    EXPERIMENT_NEW_RESPONSE_SCREEN_KEYBOARD("app_new_r
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_COMPANY("and
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_ADDRESS("and
    EXPERIMENT_ANDROID_MORE_ADS_DIRECT("android_more_a
    EXPERIMENT_ANDROID_RENAME_RESPONSE_TO_WANT_TO_WORK
    EXPERIMENT_ANDROID_PROFILE_PROGRESS_BAR("android p
    EXPERIMENT_ANDROID_AUTO_OPEN_EDIT_WHEN_NO_RESUME("
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUT
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUT
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUT
    EXPERIMENT_ANDROID_PROFILE_REPLACE_RESUME_BUILDER(
    EXPERIMENT_ANDROID_SHOW_RESUME_STATISTICS("android
}
```

```
 */
enum class ApplicantExperiments(
        val key: String,
        val description: String = String.EMPTY
) {
    EXPERIMENT_APPS_COMPLETION_ON_RESPONSE("apps complet
    EXPERIMENT_APPS_CLUSTERS_ON_JOBSEARCH("apps clusters
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_CLUSTERS("andro
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_ADVANCED_SEARCH
    EXPERIMENT_ANDROID_FILTERS_AND_ADVANCED_SEARCH("andr
    EXPERIMENT_ANDROID_WANT_TO_WORK_AND_SUBSCRIBE("andro
    EXPERIMENT_ANDROID_ANDROID_WANT_TO_WORK_ONLY("androi
    EXPERIMENT_APPS_REGION_ON_MAIN_SCREEN("apps_region_o
    EXPERIMENT_APPS_NATIVE_OK_LOGIN("apps_native_ok_logi
    EXPERIMENT_APPS_NATIVE_VK_LOGIN("apps_native_vk_logi
    EXPERIMENT_APPS_CHANGE_POSITION_FROM_TITLE("apps_cha
    EXPERIMENT_NEW_RESPONSE_SCREEN("app_new_response_scr
    EXPERIMENT_NEW_RESPONSE_SCREEN_KEYBOARD("app_new_res
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_COMPANY("andro
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_ADDRESS("andro
    EXPERIMENT_ANDROID_MORE_ADS_DIRECT("android_more_ads
    EXPERIMENT_ANDROID_RENAME_RESPONSE_TO_WANT_TO_WORK("
    EXPERIMENT_ANDROID_PROFILE_PROGRESS_BAR("android pro
    EXPERIMENT_ANDROID_AUTO_OPEN_EDIT_WHEN_NO_RESUME("au
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTLI
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTLI
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTLI
    EXPERIMENT_GH_ANDROID_MAIN_REMOVE_TITLE("gh_android_
    EXPERIMENT_GH_ANDROID_MAIN_INCREASE_VACANCY_COUNT("g
    EXPERIMENT_GH_ANDROID_MAIN_REMINDER("gh_android_main
    EXPERIMENT_ANDROID_PROFILE_REPLACE_RESUME_BUILDER("a
    EXPERIMENT_ANDROID_SHOW_RESUME_STATISTICS("android_s
}
```

```
 */
enum class ApplicantExperiments(
        val key: String,
        val description: String = String.EMPTY
) {
    EXPERIMENT_APPS_COMPLETION_ON_RESPONSE("apps complet
    EXPERIMENT_APPS_CLUSTERS_ON_JOBSEARCH("apps clusters
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_CLUSTERS("andro
    EXPERIMENT_ANDROID_QUICK_FILTERS_AND_ADVANCED_SEARCH
    EXPERIMENT_ANDROID_FILTERS_AND_ADVANCED_SEARCH("andr
    EXPERIMENT_ANDROID_WANT_TO_WORK_AND_SUBSCRIBE("andro
    EXPERIMENT_ANDROID_ANDROID_WANT_TO_WORK_ONLY("androi
    EXPERIMENT_APPS_REGION_ON_MAIN_SCREEN("apps_region_o
    EXPERIMENT_APPS_NATIVE_OK_LOGIN("apps_native_ok_logi
    EXPERIMENT_APPS_NATIVE_VK_LOGIN("apps_native_vk_logi
    EXPERIMENT_APPS_CHANGE_POSITION_FROM_TITLE("apps_cha
    EXPERIMENT_NEW_RESPONSE_SCREEN("app_new_response_scr
    EXPERIMENT_NEW_RESPONSE_SCREEN_KEYBOARD("app_new_res
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_COMPANY("andro
    EXPERIMENT_ANDROID_RESPONSE_BTN_AFTER_ADDRESS("andro
    EXPERIMENT_ANDROID_MORE_ADS_DIRECT("android_more_ads
    EXPERIMENT_ANDROID_RENAME_RESPONSE_TO_WANT_TO_WORK("
    EXPERIMENT_ANDROID_PROFILE_PROGRESS_BAR("android pro
    EXPERIMENT_ANDROID_PROFILE_ADVANCED_INFO("android pr
    EXPERIMENT_ANDROID_AUTO_OPEN_EDIT_WHEN_NO_RESUME("au
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTL
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTL
    EXPERIMENT_GH_ANDROID_VACANCY_LIST_BUTTON_WITH_OUTL
    EXPERIMENT_GH_ANDROID_MAIN_REMOVE_TITLE("gh_androic
    EXPERIMENT_GH_ANDROID_MAIN_INCREASE_VACANCY_COUNT('
    EXPERIMENT_GH_ANDROID_MAIN_REMINDER("gh_android_main
}
```

Три основные проблемы

1. Постоянные merge-конфликты в общем наборе экспериментов
2. **Пересборка почти всего приложения при добавлении эксперимента**
3. Ради проверки эксперимента писали очень много кода

Три основные проблемы

1. Постоянные merge-конфликты в общем наборе экспериментов
2. Пересборка почти всего приложения при добавлении эксперимента
3. **Ради проверки эксперимента писали очень много кода**

Театр начинается с feature-модуля

```
// :features:feature-module → FeatureModuleDependencies.kt

interface FeatureModuleDependencies {

    fun isFirstExperimentEnabled(): Boolean
    fun isSecondExperimentEnabled(): Boolean

}
```

Переходим в app-медиатор

```
// :app → FeatureModuleMediator.kt
class FeatureModuleDependenciesImpl : FeatureModuleDependencies {

    override fun isFirstExperimentEnabled(): Boolean =
        getExperimentsConfig().isFirstExperimentEnabled()

    override fun isSecondExperimentEnabled(): Boolean =
        getExperimentsConfig().isSecondExperimentEnabled()

    private fun getExperimentsConfig(): ExperimentsConfig =
        DI.getAppScope().getInstance(ExperimentsConfig::class.java)

}
```

Оттуда — в модуль экспериментов

```
// :core:experiments:appicant → ApplicantExperimentsConfig.kt

interface ApplicantExperimentsConfig {

    fun isFirstExperimentEnabled(): Boolean
    fun isSecondExperimentEnabled(): Boolean

}
```

Реализация конфига

```
// :core:experiments:appicant → ApplicantExperimentsConfig.kt

internal class ApplicantExperimentsConfigImpl @Inject constructor(
    private val experimentsInteractor: ExperimentsInteractor
) {

    override fun isFirstExperimentEnabled(): Boolean = experimentsInteractor.isUserAffected(
        ApplicantExperiments.FIRST_EXPERIMENT.key
    )

    override fun isSecondExperimentEnabled(): Boolean = experimentsInteractor.isUserAffected(
        ApplicantExperiments.SECOND_EXPERIMENT.key
    )

}
```

И, наконец, enum с экспериментами

```
// :core:experiments:appicant → ApplicantExperimentsConfig.kt

internal enum class ApplicantExperiments(
    val key: String
) {

    FIRST_EXPERIMENT("first_key"),
    SECOND_EXPERIMENT("second_key"),

}
```

Выводы

1. Нам нужны эксперименты в кодовой базе
2. Тратим время на merge-конфлиты
3. Тратим время на проброс флагов


Уходим от
merge conflicts

Что можно сделать

1. Добавляем интерфейс Experiment
2. Каждый эксперимент — отдельный класс
3. Для проверки вхождения — статический метод

Добавляем интерфейс Experiment

```
// :core:experiments → Experiment.kt

interface Experiment {
    val key: String
}
```

Каждый эксперимент — класс

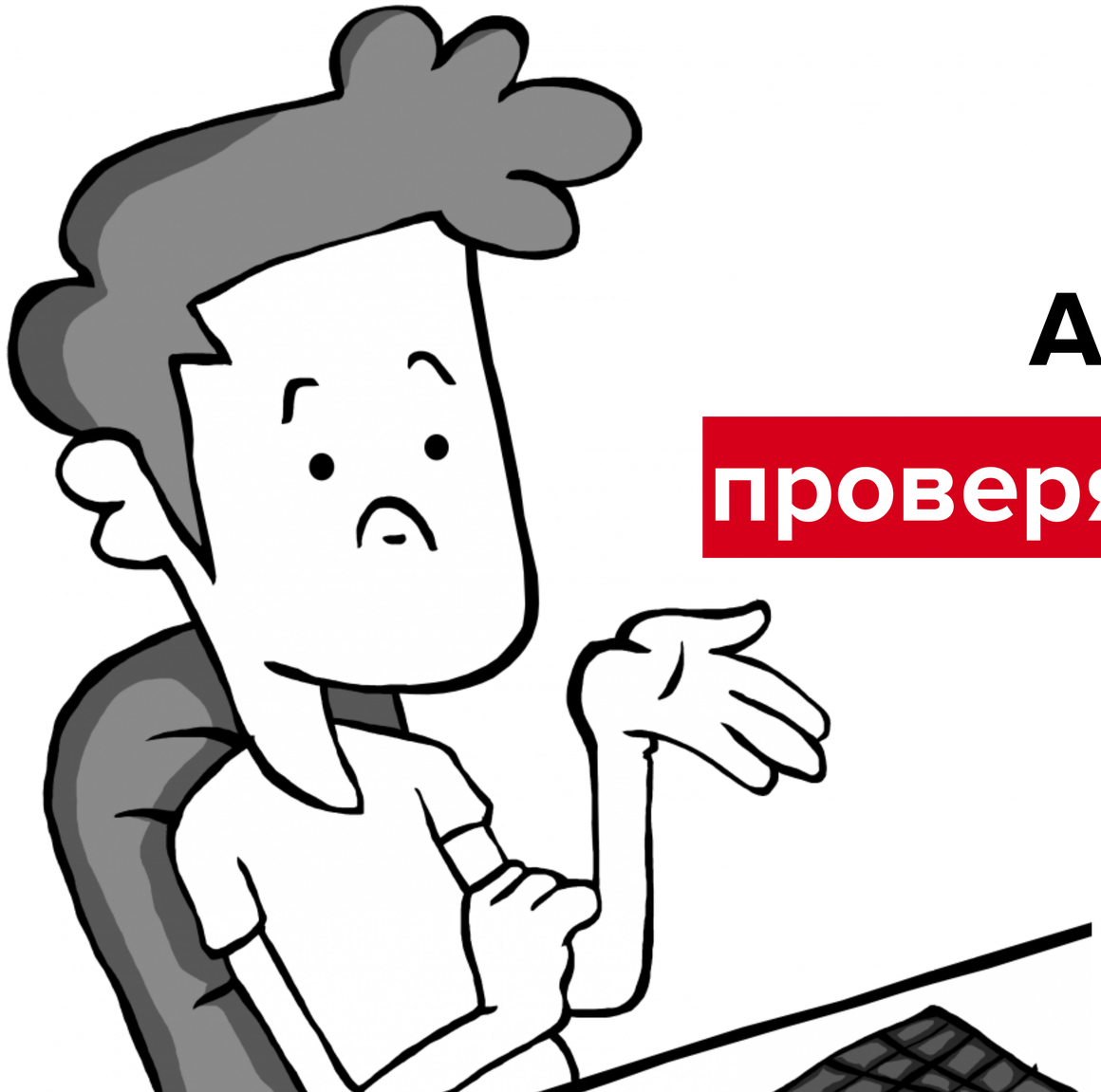
```kotlin
// :features:first → FirstFeatureExperiment.kt

internal class FirstFeatureExperiment : Experiment {
    override val key: String get() = "first_key"
}

// :features:second → SecondFeatureExperiment.kt

internal class SecondFeatureExperiment : Experiment {
    override val key: String get() = "second_key"
}

// :features:third → ThirdFeatureExperiment.kt

internal class ThirdFeatureExperiment : Experiment {
    override val key: String get() = "third_key"
}
```

Статический метод проверки

```
// :core:experiments → Experiments.kt

object Experiments {

    private var experimentsInteractor: ExperimentsInteractor? = null

    fun init(experimentsInteractor: ExperimentsInteractor) {
        this.experimentsInteractor = experimentsInteractor
    }

    fun isUserAffected(experiment: Experiment): Boolean {
        return experimentsInteractor?.isUserAffected(experimentKey = experiment.key)
            ?: false
    }
}
```

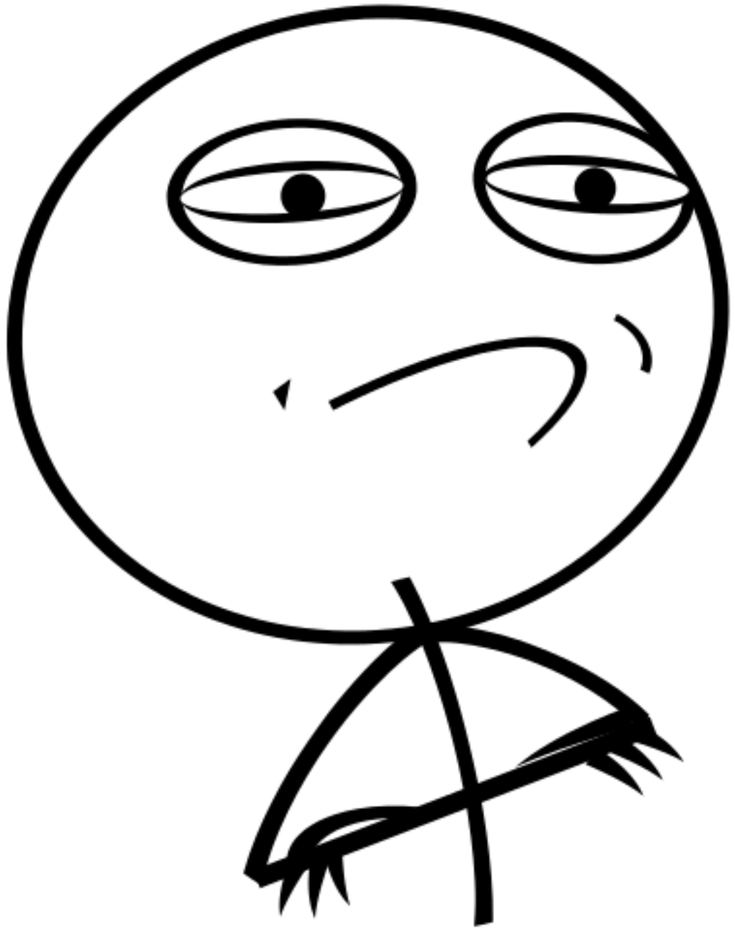
Для удобства — extension-метод

```
// :core:experiments → ExperimentsExt.kt

fun Experiment.isUserAffected(): Boolean =
    Experiments.isUserAffected(this)


// :features:first → somewhere

FirstFeatureExperiment().isUserAffected()
```



А если эксперимент

проверяется в разных модулях?

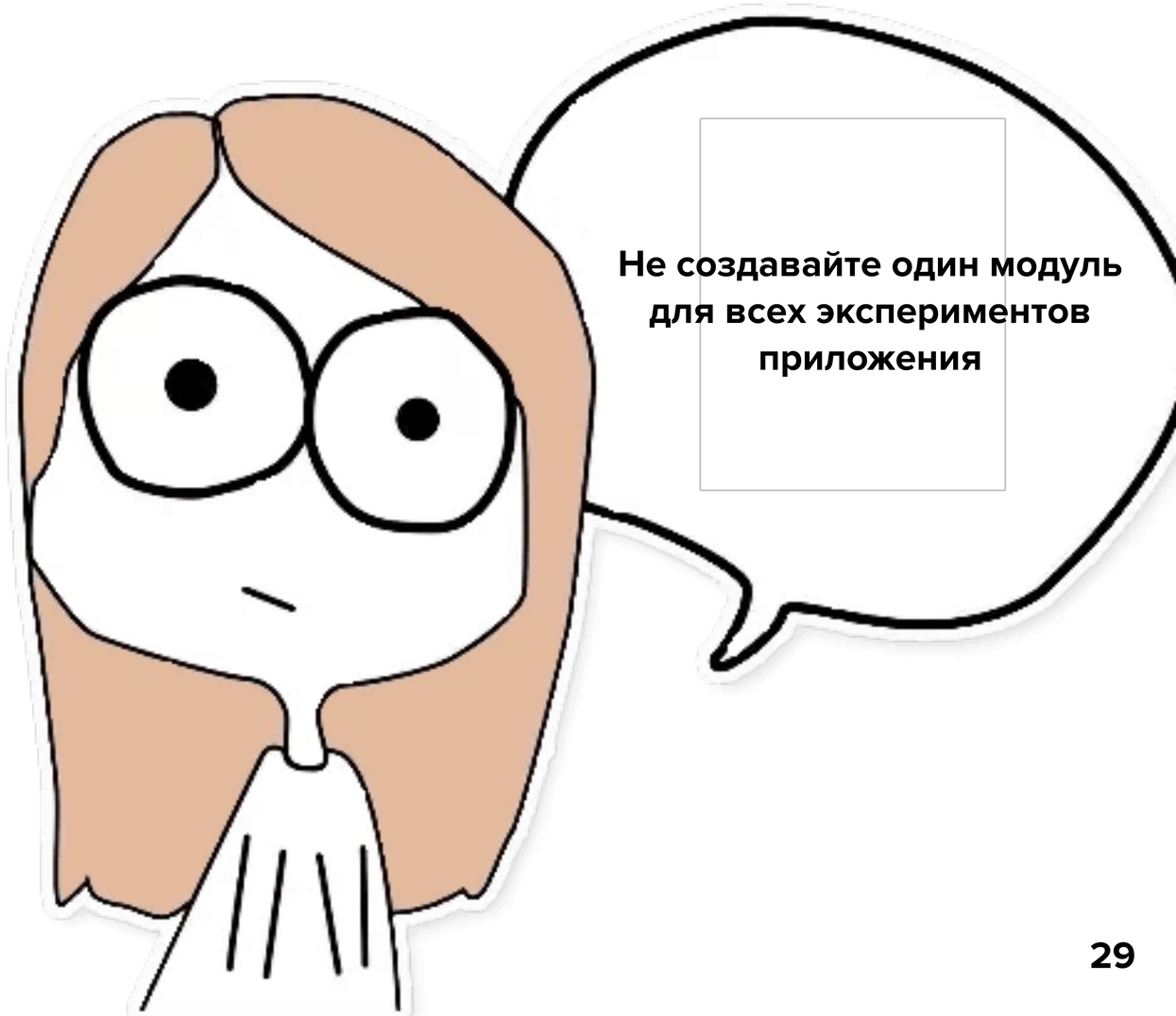


Выносите модель

в общий модуль

Используем общий модуль

```
// :core:experiments:common → CommonExperiment.kt

class CommonExperiment : Experiment {
    override val key: String get() = "common_key"
}

// :features:first → somewhere

CommonExperiment.isUserAffected()

// :features:second → somewhere

CommonExperiment.isUserAffected()
```



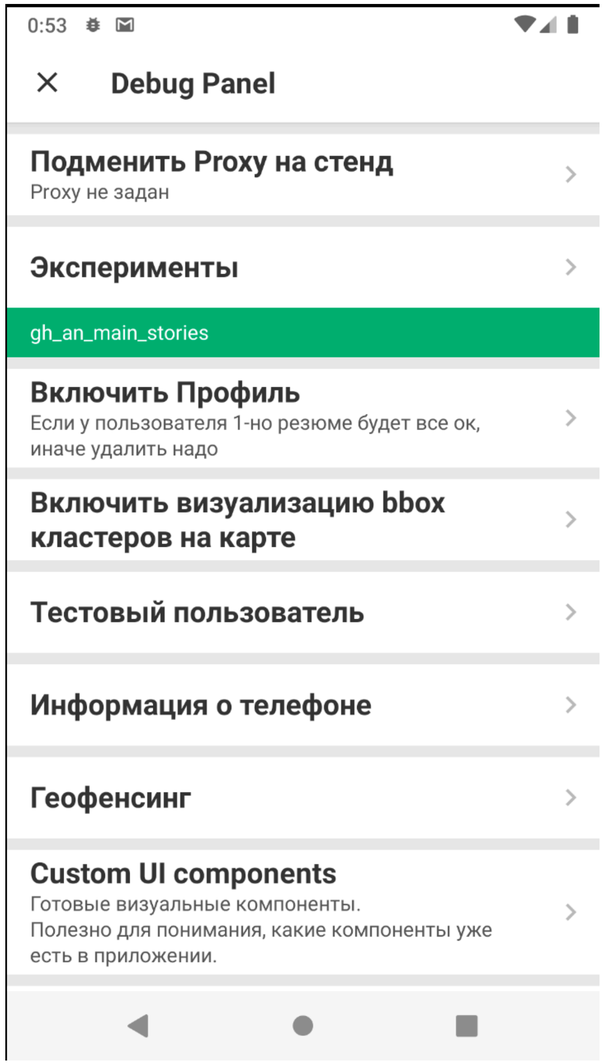
Выводы

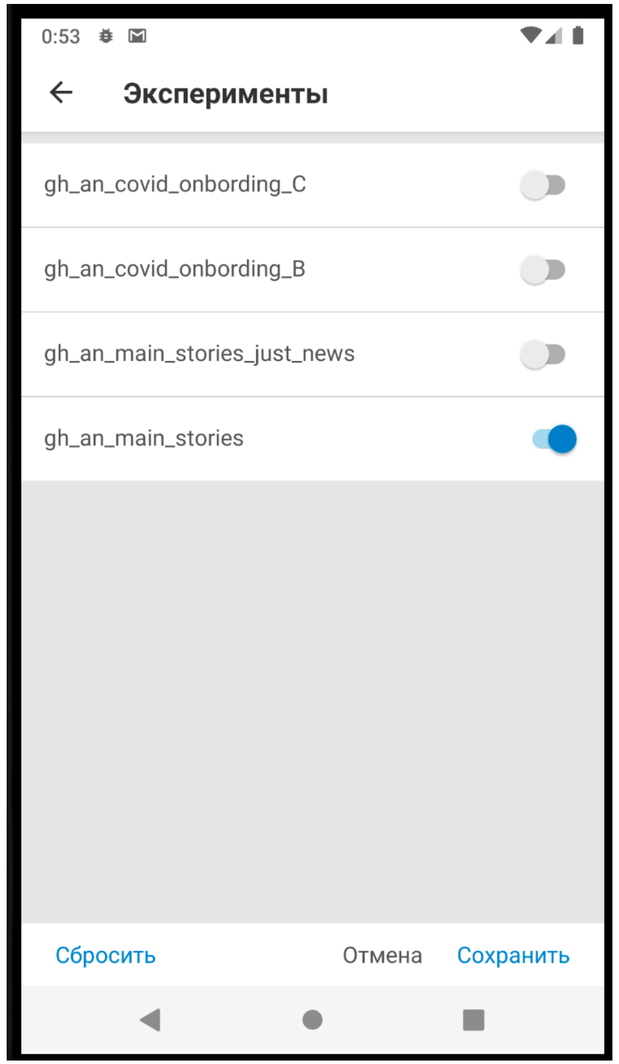
1. Можно уйти от merge-конфликтов
2. Можно уйти от пересборки множества модулей
3. Схожая техника поможет и в других ситуациях


Собираем
feature флаги

В чём вообще проблема?

У нас есть дебаг-панель



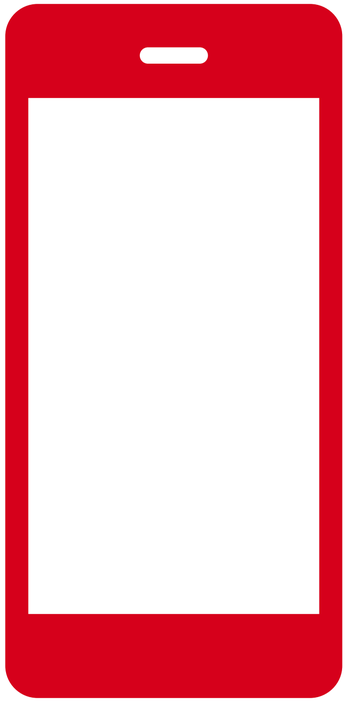


От enum-чика ушёл...

```
enum class ApplicantExperiments(val key: String) {
    EXPERIMENT_ONE("exp_one"),
    EXPERIMENT_TWO("exp_two"),
    EXPERIMENT_THREE("exp_three"),
    EXPERIMENT_FOUR("exp_four"),
    EXPERIMENT_FIVE("exp_five"),
    EXPERIMENT_SIX("exp_six"),
    EXPERIMENT_SEVEN("exp_seven"),
    EXPERIMENT_EIGHT("exp_eight"),
    EXPERIMENT_NINE("exp_nine"),
    EXPERIMENT_TEN("exp_ten"),
    EXPERIMENT_ELEVEN("exp_eleven"),
    EXPERIMENT_TWELVE("exp_twelve"),
    EXPERIMENT_THIRTEEN("exp_thirteen"),
    ...
}
```

```
// :features:first
class ExperimentOne : Experiment {
    override val key: String get() = "exp_one"
}

// :features:second
class ExperimentTwo : Experiment {
    override val key: String get() = "exp_two"
}

// :features:third
class ExperimentThree : Experiment {
    override val key: String get() = "exp_three"
}
```

Сервер не присылает весь список



GET /experiments

```
{ "exp_one": true, "exp_three": true }
```

Как теперь

получить список?

Способы получения списка

1. Собрать вручную
2. Воспользоваться DI-фреймворком
3. Вспомнить про Java Reflection API
4. Сгенерировать нужный код

Собрать вручную?..

Добавляем фичи в дебаг-панель

```
// :debug-panel/build.gradle.kts

dependencies {
    implementation(project(":features:first"))
    implementation(project(":features:second"))
    implementation(project(":features:third"))
    ...

    // Нужно добавить все модули с моделями экспериментов
}
```

Собираем список

```
// :debug-panel/DebugPanelViewModel.kt

private fun getAllExperiments(): List<Experiment> {
    return listOf(
        FirstFeatureExperiment(),
        SecondFeatureExperiment(),
        ThirdFeatureExperiment(),
    )
}
```


Merge-конфликты
Merge-конфликты

Выводы

+ Никаких дополнительных зависимостей

+ Очень простая реализация

- Merge-конфликты

Способы получения списка

1. ~~Собрать вручную~~
2. **Воспользоваться DI-фреймворком**
3. **Вспомнить про Java Reflection API**
4. **Сгенерировать нужный код**

Возможности DI-фреймворка

01

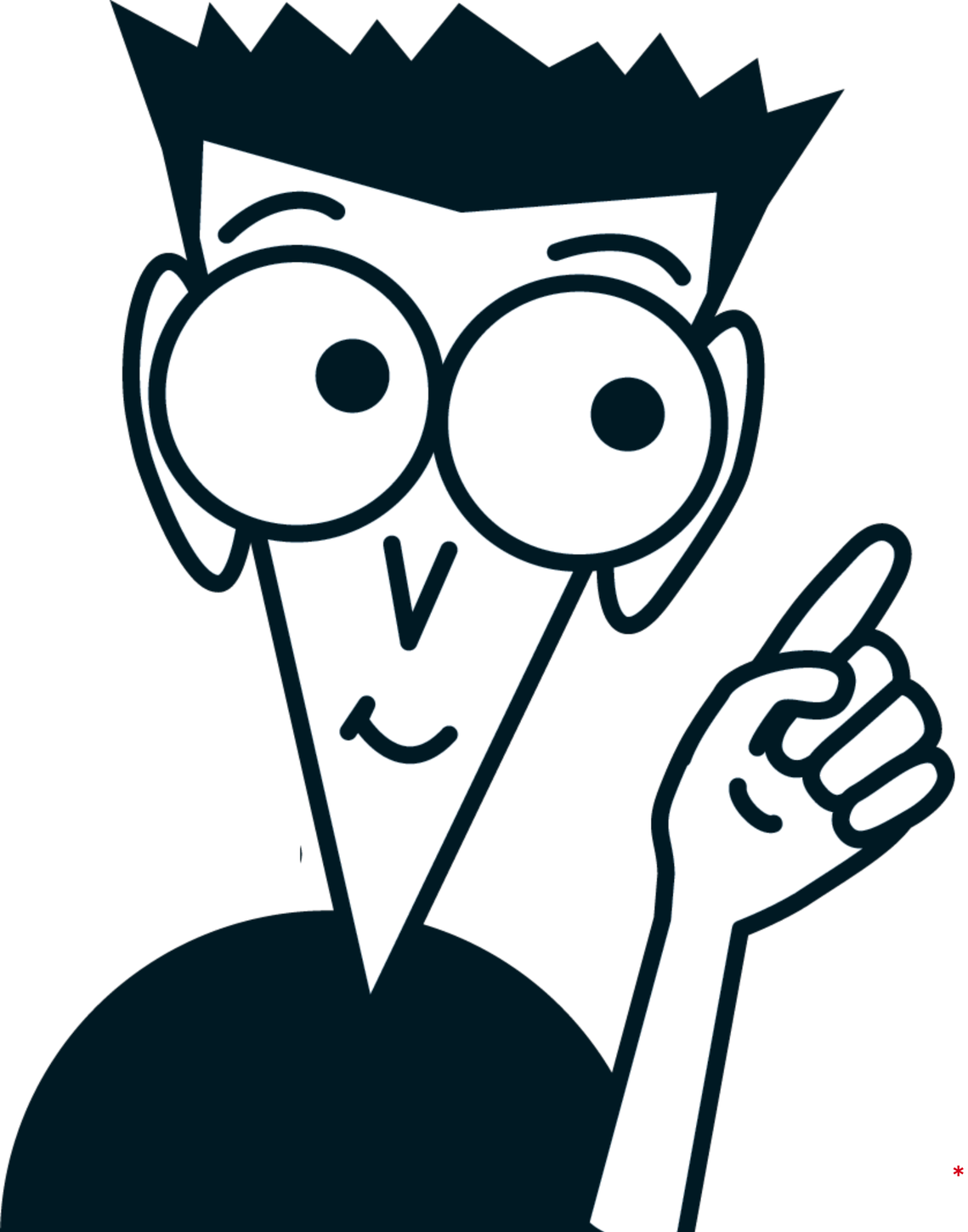
Возможности
DI-фреймворка

02

Java
Reflection API

03

Codegen /
Bytecode patching



DI-фреймворки* могут собирать объекты в списки

* — К сожалению, далеко не все DI-фреймворки умеют это делать

Dagger 2 / Hilt Multibindings

1. Есть возможность собирать объекты в Set<MyClass>

2. Или даже в Map<CustomKey, MyClass>

Собираем Set<Experiment>

```
// :features:first
@Module @InstallIn(SingletonComponent::class)
internal class ExperimentOneModule {

    @Provides @IntoSet
    fun providesExperiment(): Experiment = ExperimentOne()

}

// :features:second
@Module @InstallIn(SingletonComponent::class)
internal class ExperimentTwoModule {

    @Provides @IntoSet
    fun providesExperiment(): Experiment = ExperimentTwo()

}
```

Создаём dagger-модули

```
// :features:first
@Module @InstallIn(SingletonComponent::class)
internal class ExperimentOneModule {

    @Provides @IntoSet
    fun providesExperiment(): Experiment = ExperimentOne()

}

// :features:second
@Module @InstallIn(SingletonComponent::class)
internal class ExperimentTwoModule {

    @Provides @IntoSet
    fun providesExperiment(): Experiment = ExperimentTwo()

}
```

Добавляем модели в Set

```
// :features:first
@Module @InstallIn(SingletonComponent::class)
internal class ExperimentOneModule {

    @Provides @IntoSet
    fun providesExperiment(): Experiment = ExperimentOne()

}

// :features:second
@Module @InstallIn(SingletonComponent::class)
internal class ExperimentTwoModule {

    @Provides @IntoSet
    fun providesExperiment(): Experiment = ExperimentTwo()

}
```

Инжектим Set<Experiment>

```
// :debug-panel/DebugPanelViewModel.kt

@HiltViewModel
internal class DebugPanelViewModel @Inject constructor(
    private val applicationContext: Context,
    private val debugExperimentsInteractor: DebugExperimentsInteractor,
    private val allExperiments: Set<@JvmSuppressWildcards Experiment>
) : ViewModel() {
    ...
}
```

Инжектим Set<Experiment>

```
// :debug-panel/DebugPanelViewModel.kt

@HiltViewModel
internal class DebugPanelViewModel @Inject constructor(
    private val applicationContext: Context,
    private val debugExperimentsInteractor: DebugExperimentsInteractor,
    private val allExperiments: Set<@JvmSuppressWildcards Experiment>
) : ViewModel() {
    ...
}
```

@JvmSuppressWildcards

```
// :debug-panel/DebugPanelViewModel.kt

@HiltViewModel
internal class DebugPanelViewModel @Inject constructor(
    private val applicationContext: Context,
    private val debugExperimentsInteractor: DebugExperimentsInteractor,
    private val allExperiments: Set<@JvmSuppressWildcards Experiment>
) : ViewModel() {
    ...
}
```

А что там у Toothpick / Koin?

Ничего ;(

InsertKoinIO / koin

Code | Issues 106 | Pull requests 8 | Discussions | Actions | Projects | Wiki | Security | Insights

Multibinding like in Dagger 2 #772

Closed **kirich1409** opened this issue on 8 Apr 2020 · 7 comments

kirich1409 commented on 8 Apr 2020 — Contributor

Dagger 2 has a good feature "Multibinding". It allows injection of multiple dependencies into collection. It's useful to make access by a key in a map of objects to make pluggable architecture.

👍 2

stephanenicolas / toothpick

Code | Issues 38 | Pull requests 3 | Actions | Projects | Wiki | Security | Insights

Dagger2 MultiBindings #368

Open **adolgiy** opened this issue on 16 Sep 2019 · 14 comments

adolgiy commented on 16 Sep 2019 • edited

Would be very happy to have something like Dagger2 MultiBindings in Toothpick! IntoSet/IntoMap sometimes are very useful.

Выводы

\+ Доступно из коробки в Dagger 2 / Hilt

\- Доступно только в Dagger 2 / Hilt

Java Reflection API

01

Возможности
DI-фреймворка

02

Java
Reflection API

03

Codegen /
Bytecode patching

Reflection API

1. **ServiceLoader + META-INF/services**
2. ClassLoader + DexFile
3. Scannotation / Reflections / Classgraph / etc

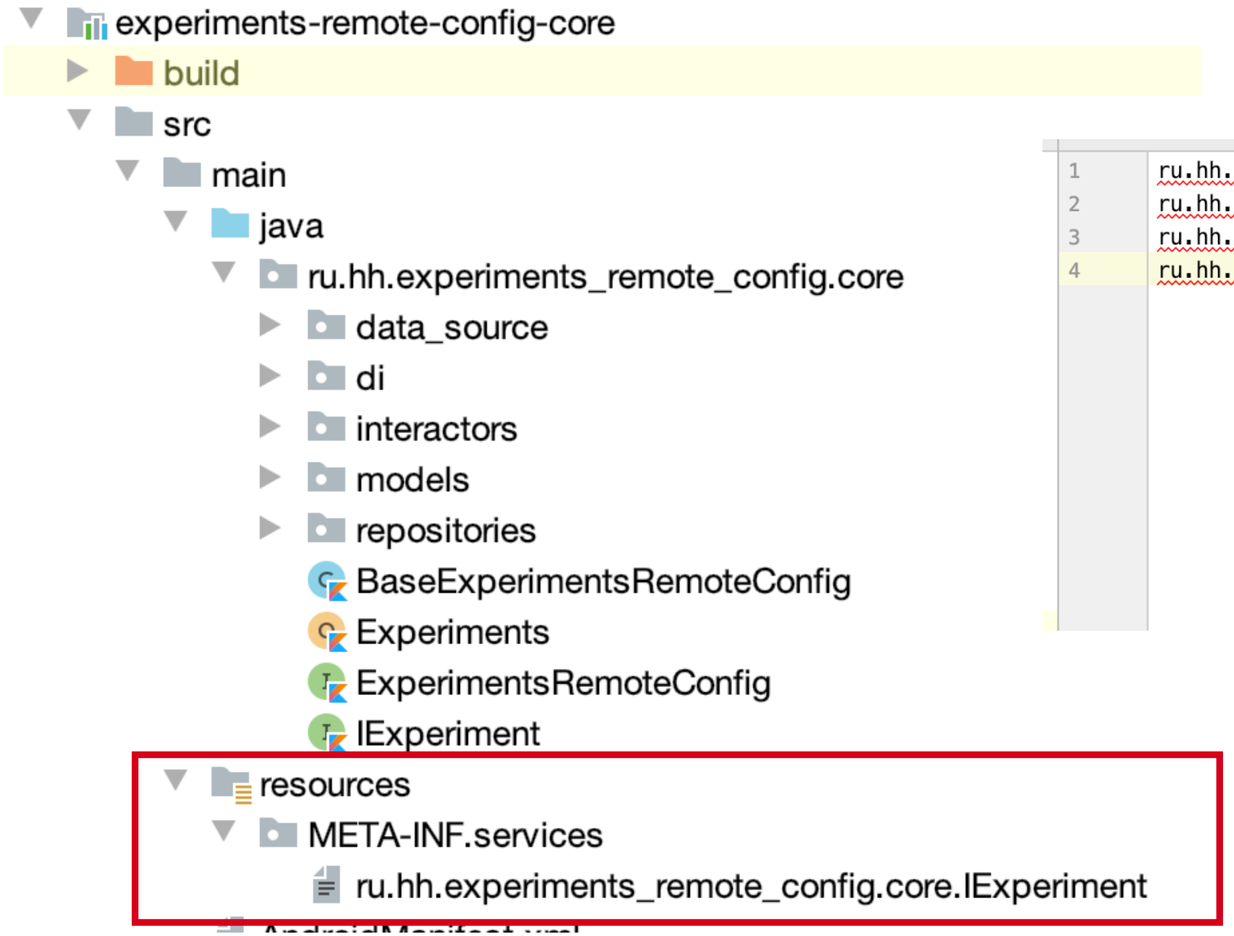
ServiceLoader

ServiceLoader

1. Создать файл /src/resources/META-INF/services/fqcn.Experiment

2. ServiceLoader.load(Experiment::class.java)

META-INF/services/FQCN.Experiment




```
1  ru.hh.feature_intentions_onboarding.experiments.GhAndroidHintWorkOnboardingOldQuickQueryTagsExperiment
2  ru.hh.feature_intentions_onboarding.experiments.GhAndroidHintWorkOnboardingNewQuickQueryTagsExperiment
3  ru.hh.feature_search.applicant.history.main.experiments.GhShowStoriesExperiment
4  ru.hh.feature_search.applicant.history.main.experiments.GhMainStoriesJustNewsExperiment
```

На каждой строчке указываем FQCN класса-реализации Experiment

Не важно в каком модуле вы создаёте этот файл

Получаем список моделек

```
// :debug-panel/DebugPanelViewModel.kt

fun getAllExperiments(): List<Experiment> {
    return ServiceLoader.load(Experiment::class.java).toList()
}
```

И вроде всё хорошо, но...

```
1  ru.hh.feature_intentions_onboarding.experiments.GhAndroidHintWorkOnboardingOldQuickQueryTagsExperiment
2  ru.hh.feature_intentions_onboarding.experiments.GhAndroidHintWorkOnboardingNewQuickQueryTagsExperiment
3  ru.hh.feature_search.applicant.history.main.experiments.GhShowStoriesExperiment
4  ru.hh.feature_search.applicant.history.main.experiments.GhMainStoriesJustNewsExperiment
```


Merge-конфликты
Merge-конфликты

Нет поддержки авторефакторинга

```
package ru.hh.android.features.first.diff_package

import ru.hh.android.core.experiments.models.Experiment


internal class AnotherExperimentName : Experiment {
```

Default Changelist

```
internal class FirstFeatureExperiment : Experiment {
}
```

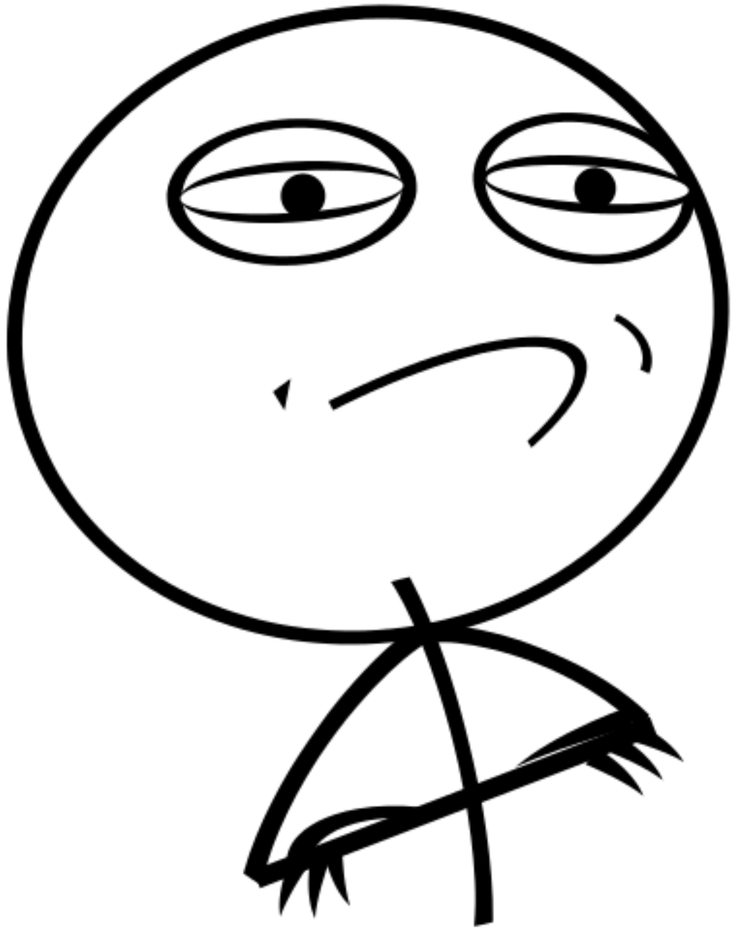
```
ru.hh.android.features.first.experiment.FirstFeatureExperiment
ru.hh.android.features.second.experiment.SecondFeatureExperiment
ru.hh.android.features.third.experiment.ThirdFeatureExperiment
```

Выводы

+ Никаких дополнительных зависимостей

+ Простое решение

- Проблема ручного заполнения META-INF/services



Сгенерируем

META-INF/services

Есть несколько готовых библиотек

1. metainf-services
2. ClassIndex

metainf-services — немного устарела

```
> Task :features:second:kaptDebugKotlin

warning: No SupportedSourceVersion annotation found on
  org.kohsuke.metainf_services.AnnotationProcessorImpl, returning RELEASE_6.warning:
    Supported source version 'RELEASE_6' from annotation processor
    'org.jetbrains.kotlin.kapt3.base.ProcessorWrapper' less than -source ‘1.8’
Note: Writing META-INF/services/ru.hh.android.core.experiments.models.Experiment

> Task :features:first:kaptDebugKotlin

warning: No SupportedSourceVersion annotation found on
  org.kohsuke.metainf_services.AnnotationProcessorImpl, returning RELEASE_6.warning:
    Supported source version 'RELEASE_6' from annotation processor
    'org.jetbrains.kotlin.kapt3.base.ProcessorWrapper' less than -source ‘1.8’
Note: Writing META-INF/services/ru.hh.android.core.experiments.models.Experiment

> Task :app:kaptDebugKotlin

warning: No SupportedSourceVersion annotation found on
  org.kohsuke.metainf_services.AnnotationProcessorImpl, returning RELEASE_6.warning:
    Supported source version 'RELEASE_6' from annotation processor
    'org.jetbrains.kotlin.kapt3.base.ProcessorWrapper' less than -source '1.8'
```

ClassIndex

1. Подключаем через compileOnly + kapt
2. Навешиваем аннотацию @IndexSubclasses на интерфейс Experiment
3. Подключаем библиотеку в фиче-модули, где есть модельки

Индексируем базовый интерфейс

```
// :core:experiments → build.gradle.kts

dependencies {
    compileOnly("org.atteo.classindex:classindex:3.4")
    kapt("org.atteo.classindex:classindex:3.4")
    ...
}

// :core:experiments → Experiment.kt

@org.atteo.classindex.IndexSubclasses
interface Experiment {
    val key: String
}
```

Подключаем ClassIndex к фичам

```
// :features:first → build.gradle.kts

dependencies {
    compileOnly("org.atteo.classindex:classindex:3.4")
    kapt("org.atteo.classindex:classindex:3.4")
    ...
}

// :features:second → build.gradle.kts

dependencies {
    compileOnly("org.atteo.classindex:classindex:3.4")
    kapt("org.atteo.classindex:classindex:3.4")
    ...
}
```

И снова используем ServiceLoader

```
// :debug-panel/DebugPanelViewModel.kt

fun getAllExperiments(): List<Experiment> {
    return ServiceLoader.load(Experiment::class.java).toList()
}
```

Выводы по ServiceLoader

+ ServiceLoader + META-INF — самый простой способ

+ Не требует зависимостей в runtime

+/- Требует работы kapt-а

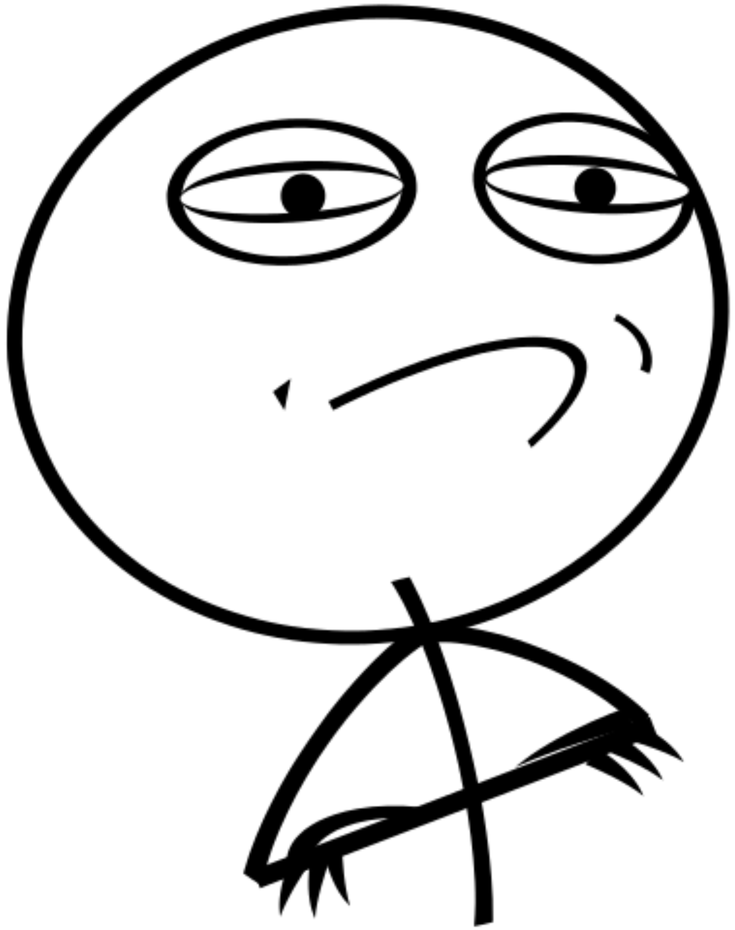
Reflection API

1. ServiceLoader + META-INF/services
2. **ClassLoader + DexFile**
3. Reflections / Scannotation / Classgraph / etc

ClassLoader + DexFile

ClassLoader + DexFile

1. Получить доступ к содержимому DexFile-а
2. Найти классы-реализации интерфейса Experiment
3. Получить список Experiment, вызывая Class.newInstance



Абстрактный сканер DexFile

Описываем сканирование DexFile-а

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {
        val classLoader = Thread.currentThread().contextClassLoader
        val dexFile = DexFile(applicationContext.packageCodePath)
        val classNamesEnumeration = dexFile.entries()

        runScanning(classNamesEnumeration)
    }

    ...
}
```

Достаём ClassLoader из Thread-а

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {
        val classLoader = Thread.currentThread().contextClassLoader
        val dexFile = DexFile(applicationContext.packageCodePath)
        val classNamesEnumeration = dexFile.entries()

        runScanning(classNamesEnumeration)
    }

    ...
}
```

Открываем DexFile

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {
        val classLoader = Thread.currentThread().contextClassLoader
        val dexFile = DexFile(applicationContext.packageCodePath)
        val classNamesEnumeration = dexFile.entries()

        runScanning(classNamesEnumeration)
    }

    ...
}
```

Проходим по всем классам в DexFile

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {...}

    private fun runScanning(classNamesEnumeration: Enumeration<String>, classLoader: ClassLoader) {
        while (classNamesEnumeration.hasMoreElements()) {
            val nextClassName = classNamesEnumeration.nextElement()
            if (isAcceptableClassName(nextClassName)) {
                val clazz = classLoader.loadClass(nextClassName)
                if (isAcceptableClass(clazz)) {
                    onScanResult(clazz)
                }
            }
        }
    }
}
```

Проходим по всем классам в DexFile

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {...}

    private fun runScanning(classNamesEnumeration: Enumeration<String>, classLoader: ClassLoader) {
        while (classNamesEnumeration.hasMoreElements()) {
            val nextClassName = classNamesEnumeration.nextElement()
            if (isAcceptableClassName(nextClassName)) {
                val clazz = classLoader.loadClass(nextClassName)
                if (isAcceptableClass(clazz)) {
                    onScanResult(clazz)
                }
            }
        }
    }
}
```

Фильтруем по имени класса

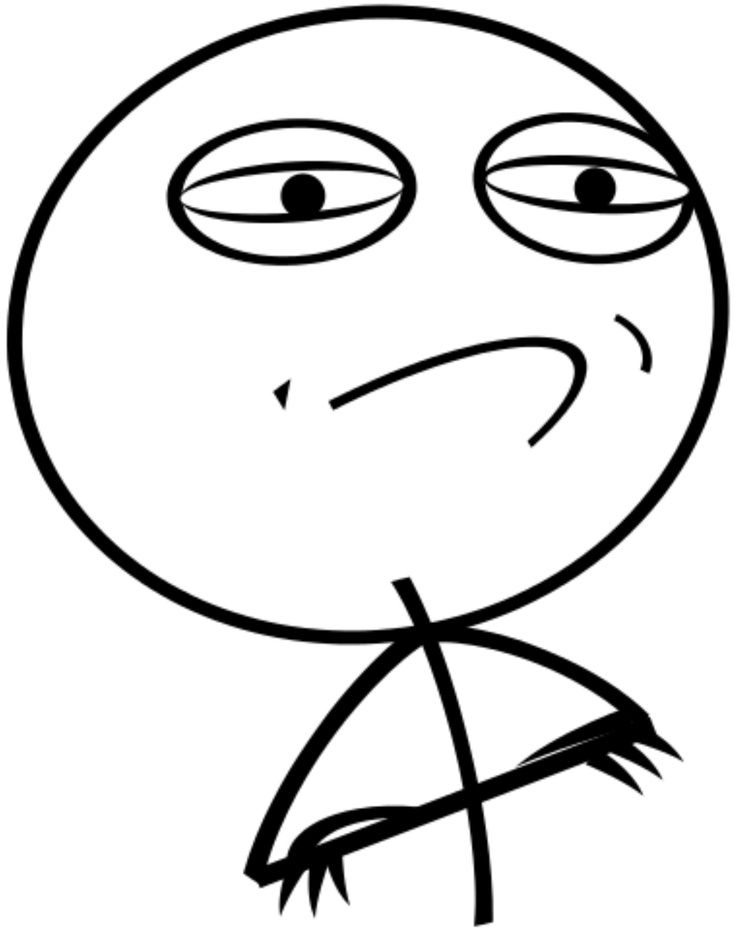
```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {...}

    private fun runScanning(classNamesEnumeration: Enumeration<String>, classLoader: ClassLoader) {
        while (classNamesEnumeration.hasMoreElements()) {
            val nextClassName = classNamesEnumeration.nextElement()
            if (isAcceptableClassName(nextClassName)) {
                val clazz = classLoader.loadClass(nextClassName)
                if (isAcceptableClass(clazz)) {
                    onScanResult(clazz)
                }
            }
        }
    }
}
```

Фильтруем по нужному интерфейсу

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {...}

    private fun runScanning(classNamesEnumeration: Enumeration<String>, classLoader: ClassLoader) {
        while (classNamesEnumeration.hasMoreElements()) {
            val nextClassName = classNamesEnumeration.nextElement()
            if (isAcceptableClassName(nextClassName)) {
                val clazz = classLoader.loadClass(nextClassName)
                if (isAcceptableClass(clazz)) {
                    onScanResult(clazz)
                }
            }
        }
    }
}
```

Аккумулируем результат

```
abstract class ClassScanner {
    protected abstract fun isAcceptableClassName(className: String): Boolean
    protected abstract fun isAcceptableClass(clazz: Class<*>): Boolean
    protected abstract fun onScanResult(clazz: Class<*>)

    fun scan(applicationContext: Context) {...}

    private fun runScanning(classNamesEnumeration: Enumeration<String>, classLoader: ClassLoader) {
        while (classNamesEnumeration.hasMoreElements()) {
            val nextClassName = classNamesEnumeration.nextElement()
            if (isAcceptableClassName(nextClassName)) {
                val clazz = classLoader.loadClass(nextClassName)
                if (isAcceptableClass(clazz)) {
                    onScanResult(clazz)
                }
            }
        }
    }
}
```



Ищем

эксперименты

Конкретная реализация сканера

```
internal class ExperimentsClassScanner : ClassScanner() {
    val experiments = mutableListOf<Experiment>()

    override fun isAcceptableClassName(className: String): Boolean {
        return className.endsWith("Experiment")
    }

    override fun isAcceptableClass(clazz: Class<*>): Boolean {
        return try {
            Experiment::class.java.isAssignableFrom(clazz) && clazz != Experiment::class.java
        } catch (ex: ClassCastException) {
            false
        }
    }

    override fun onScanResult(clazz: Class<*>) {
        experiments += clazz.newInstance() as Experiment
    }
}
```

Ищем суффикс у имени класса

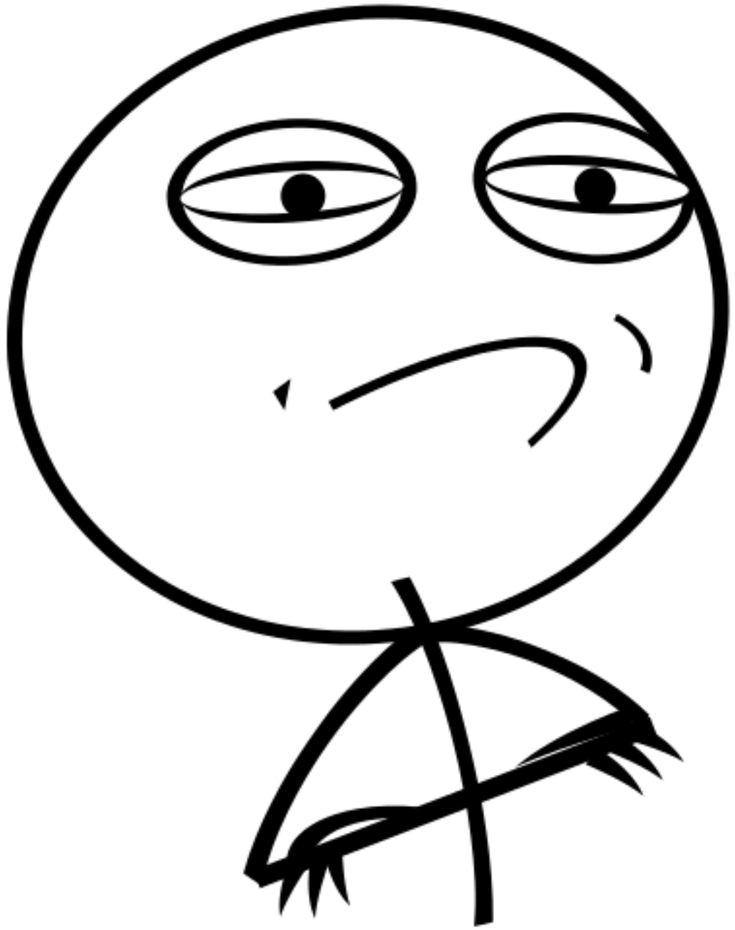
```
internal class ExperimentsClassScanner : ClassScanner() {
    val experiments = mutableListOf<Experiment>()

    override fun isAcceptableClassName(className: String): Boolean {
        return className.endsWith("Experiment")
    }

    override fun isAcceptableClass(clazz: Class<*>): Boolean {
        return try {
            Experiment::class.java.isAssignableFrom(clazz) && clazz != Experiment::class.java
        } catch (ex: ClassCastException) {
            false
        }
    }

    override fun onScanResult(clazz: Class<*>) {
        experiments += clazz.newInstance() as Experiment
    }
}
```

Ищем реализацию интерфейса

```
internal class ExperimentsClassScanner : ClassScanner() {
    val experiments = mutableListOf<Experiment>()

    override fun isAcceptableClassName(className: String): Boolean {
        return className.endsWith("Experiment")
    }

    override fun isAcceptableClass(clazz: Class<*>): Boolean {
        return try {
            Experiment::class.java.isAssignableFrom(clazz) && clazz != Experiment::class.java
        } catch (ex: ClassCastException) {
            false
        }
    }

    override fun onScanResult(clazz: Class<*>) {
        experiments += clazz.newInstance() as Experiment
    }
}
```

Складываем результат в список

```
internal class ExperimentsClassScanner : ClassScanner() {
    val experiments = mutableListOf<Experiment>()

    override fun isAcceptableClassName(className: String): Boolean {
        return className.endsWith("Experiment")
    }

    override fun isAcceptableClass(clazz: Class<*>): Boolean {
        return try {
            Experiment::class.java.isAssignableFrom(clazz) && clazz != Experiment::class.java
        } catch (ex: ClassCastException) {
            false
        }
    }

    override fun onScanResult(clazz: Class<*>) {
        experiments += clazz.newInstance() as Experiment
    }
}
```



Используем

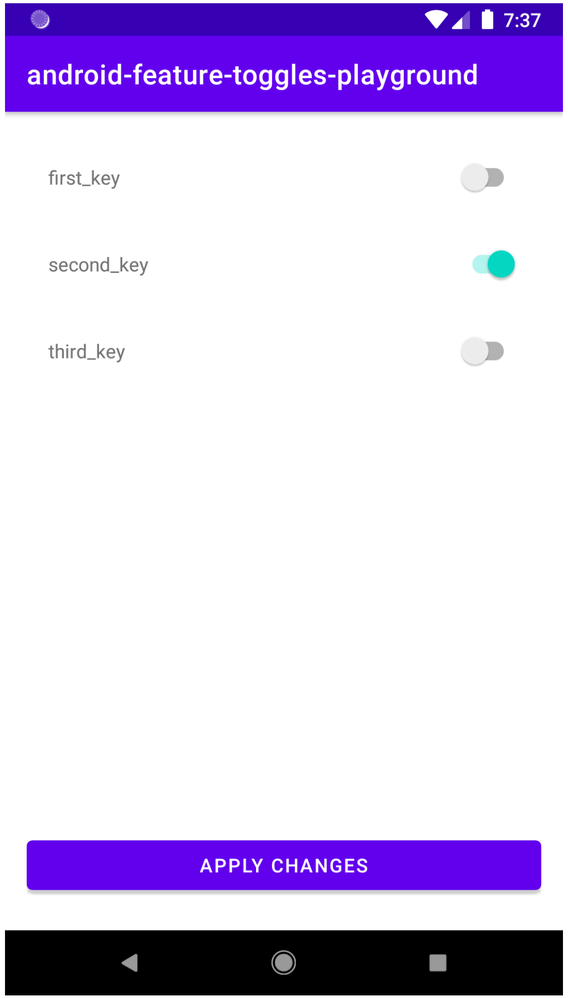
сканер

Используем созданный сканер

```
// :debug-panel → DebugPanelViewModel.kt

fun getAllExperiments(): List<Experiment> {
    return ExperimentsScanner().run {
        scan(applicationContext)
        experiments
    }
}
```

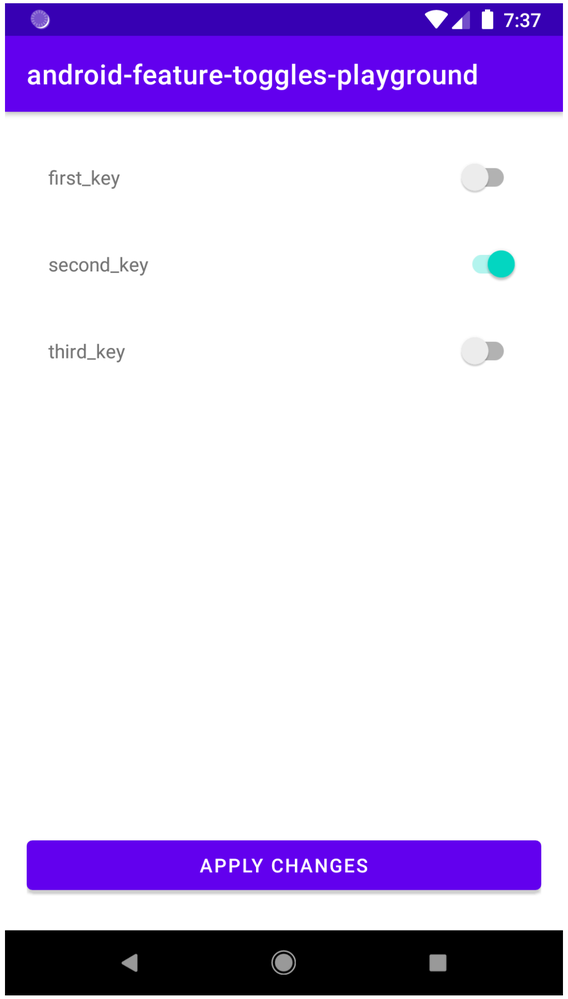
А что будет, если включить proguard?

Включаем Proguard и...




```
buildTypes {
    named("release") {
        isMinifyEnabled = true
    }
}
```

... и грустим




```
buildTypes {
    named("release") {
        isMinifyEnabled = true
    }
}
```

Дополняем proguard-rules.pro

```
// :app → proguard-rules.pro

# Keep names of every ‘Experiment' implementation
-keepnames class * implements ru.hh.android.core.experiments.models.Experiment
```

Выводы по ClassLoader + DexFile

- \+ Рабочий способ, несмотря на deprecated DexFile
- \- Требует соглашения по неймингу фиче-флагов
- \- Для minified-сборок нужно включить -keep имён
- \- Через год не поймём, зачем выбрали этот способ

Reflection API

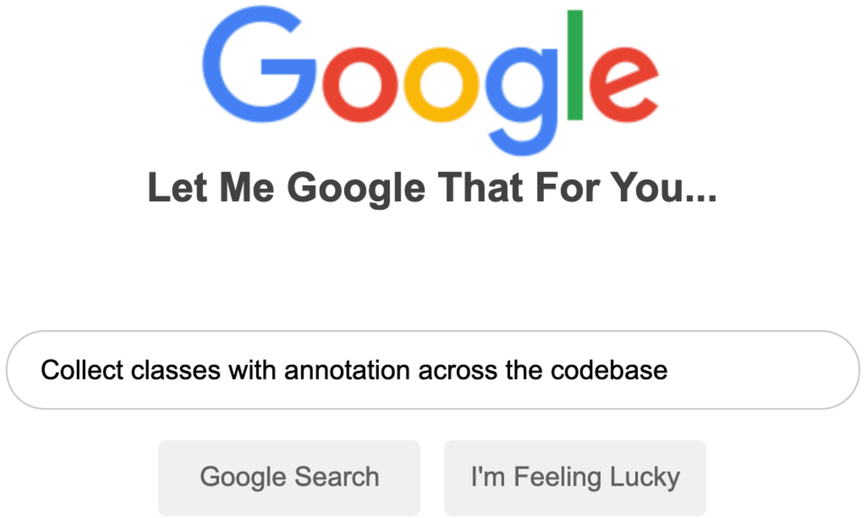
1. ServiceLoader + META-INF/services
2. ClassLoader + DexFile
3. **Reflections / Scannotation / Classgraph / etc**

Разные библиотеки

Alternatives

Some other classpath scanning mechanisms include:

- Reflections
- Corn Classpath Scanner
- annotation-detector
- Scannotation
- Sclasner
- Annovention
- ClassIndex (compiletime annotation scanner/processor)
- Jandex (Java annotation indexer, part of Wildfly)
- Spring has built-in classpath scanning
- Hibernate has the class `org.hibernate.ejb.packaging.Scanner`.
- extcos -- the Extended Component Scanner
- Javassist
- ObjectWeb ASM
- QDox, a fast Java source parser and indexer
- bndtools, which is able to "crawl"/parse the bytecode of class files to find all imports/dependencies, among other things.
- coffea, a command line tool and Python library for analyzing static dependences in Java bytecode
- org.clapper.classutil.ClassFinder
- com.google.common.reflect.ClassPath
- jdependency



Библиотеки Java Reflection API

1. **Reflections**
2. Classgraph

Библиотеки Java Reflection API

1. Reflections
2. **Classgraph**

Сэкономлю вам время

- Из коробки не работают под Android
- Чётких инструкций по этим библиотекам нет
- Нужно писать нетривиальные Gradle-скрипты

Codegen / Bytecode patching

01 Возможности DI-фреймворка

02 Java Reflection API

03 Codegen / Bytecode patching

Возможности кодогенерации

1. Можно написать свой annotation processor
2. А можно модифицировать байт-код

Annotation processor-ы уже есть

1. ~~Можно написать свой annotation processor~~

2. **А можно модифицировать байт-код**

Colonist

Colonist

1. Объявить аннотацию «колонию»
2. Описать правила, по которым «колония» ищет «поселенцев»
3. Подключить библиотеку к нужным модулям

Подключаем Colonist к приложению

```
// rootProject → build.gradle.kts
buildscript {
    dependencies {
        classpath("com.joom.colonist:colonist-gradle-plugin:0.1.0-alpha9")
    }
}

// :app → build.gradle.kts
plugins {
    id("com.android.application")
    kotlin("android")
    id("com.joom.colonist.android")
}

// :debug-panel → build.gradle.kts
dependencies {
    // Потому что в library-модули нельзя подключить плагин colonist-a
    compileOnly("com.joom.colonist:colonist-core:0.1.0-alpha9")
}
```

Описываем «колонию»

```
// :debug-panel → ExperimentsColony.kt

@Colony
@SelectSettlersBySuperType(Experiment::class)
@ProduceSettlersViaConstructor
@AcceptSettlersViaCallback
@Target(AnnotationTarget.CLASS)
internal annotation class ExperimentsColony
```

Выбираем «поселенцев» по классу

```
// :debug-panel → ExperimentsColony.kt

@Colony
@SelectSettlersBySuperType(Experiment::class)
@ProduceSettlersViaConstructor
@AcceptSettlersViaCallback
@Target(AnnotationTarget.CLASS)
internal annotation class ExperimentsColony
```

Создаём модели через конструктор

```
// :debug-panel → ExperimentsColony.kt

@Colony
@SelectSettlersBySuperType(Experiment::class)
@ProduceSettlersViaConstructor
@AcceptSettlersViaCallback
@Target(AnnotationTarget.CLASS)
internal annotation class ExperimentsColony
```

Обрабатываем в callback-е

```
// :debug-panel → ExperimentsColony.kt

@Colony
@SelectSettlersBySuperType(Experiment::class)
@ProduceSettlersViaConstructor
@AcceptSettlersViaCallback
@Target(AnnotationTarget.CLASS)
internal annotation class ExperimentsColony
```

Создаём класс-«колонию»

```
// :debug-panel → ExperimentsCollector.kt

@ExperimentsColony
internal class ExperimentsCollector {

    private val experiments = mutableListOf<Experiment>()

    init {
        Colonist.settle(this)
    }

    @OnAcceptSettler(colonyAnnotation = ExperimentsColony::class)
    fun onAcceptExperiment(experiment: Experiment) {
        experiments += experiment
    }

    fun getAllExperiments(): List<Experiment> = experiments

}
```

Пометили аннотацией «колонии»

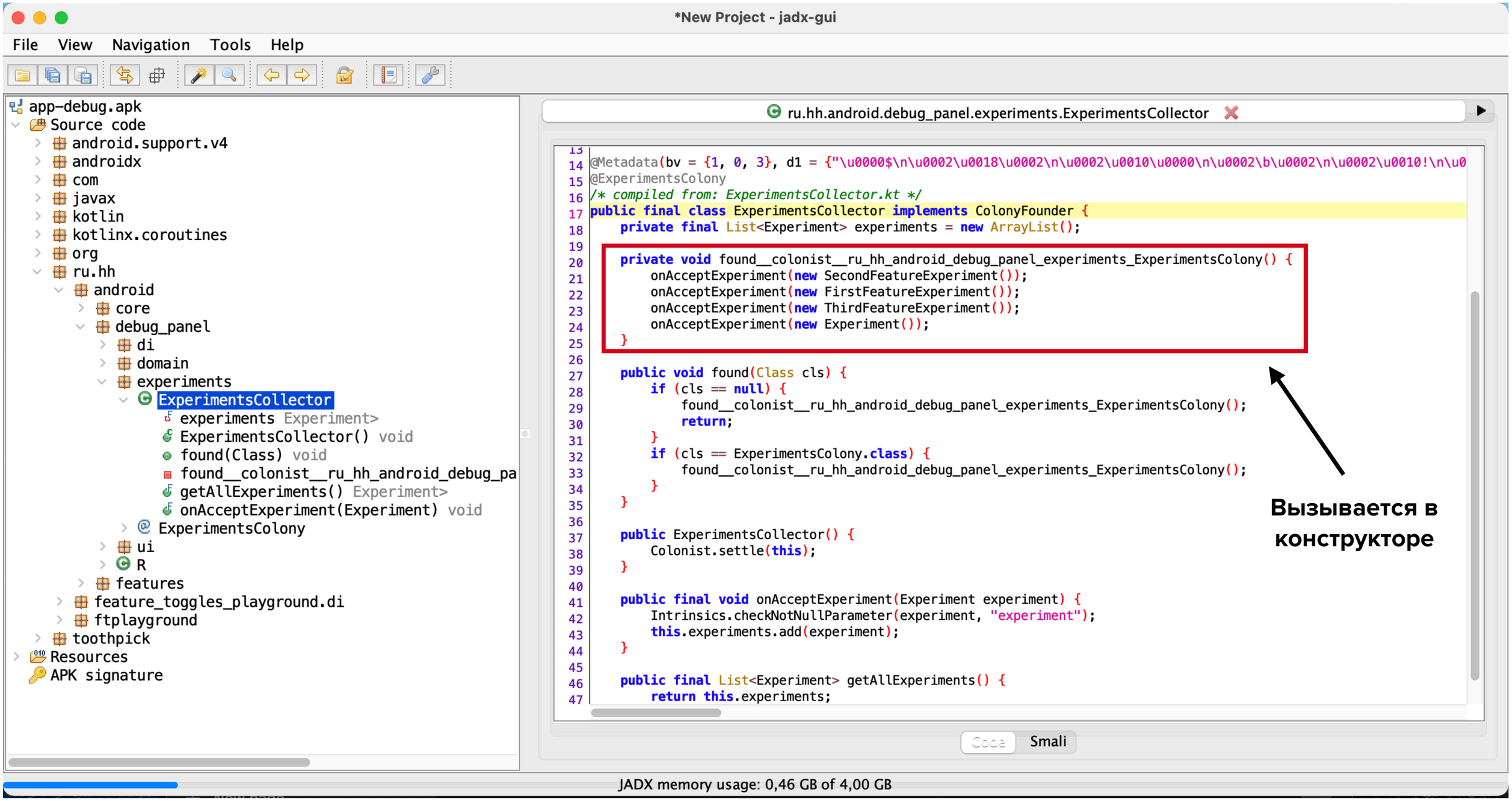
```
// :debug-panel → ExperimentsCollector.kt

@ExperimentsColony
internal class ExperimentsCollector {

    private val experiments = mutableListOf<Experiment>()

    init {
        Colonist.settle(this)
    }

    @OnAcceptSettler(colonyAnnotation = ExperimentsColony::class)
    fun onAcceptExperiment(experiment: Experiment) {
        experiments += experiment
    }

    fun getAllExperiments(): List<Experiment> = experiments

}
```

Описали добавление «поселенцев»

```
// :debug-panel → ExperimentsCollector.kt

@ExperimentsColony
internal class ExperimentsCollector {

    private val experiments = mutableListOf<Experiment>()

    init {
        Colonist.settle(this)
    }

    @OnAcceptSettler(colonyAnnotation = ExperimentsColony::class)
    fun onAcceptExperiment(experiment: Experiment) {
        experiments += experiment
    }

    fun getAllExperiments(): List<Experiment> = experiments

}
```

Запустили процесс «заселения»

```
// :debug-panel → ExperimentsCollector.kt

@ExperimentsColony
internal class ExperimentsCollector {

    private val experiments = mutableListOf<Experiment>()

    init {
        Colonist.settle(this)
    }

    @OnAcceptSettler(colonyAnnotation = ExperimentsColony::class)
    fun onAcceptExperiment(experiment: Experiment) {
        experiments += experiment
    }

    fun getAllExperiments(): List<Experiment> = experiments

}
```


*New Project - jadx-gui
File View Navigation Tools Help
app-debug.apk
Source code
android.support.v4
androidx
com
javax
kotlin
kotlinx.coroutines
org
ru.hh
android
core
debug_panel
di
domain
experiments
ExperimentsCollector
experiments Experiment>
ExperimentsCollector() void
found(Class) void
found__colonist__ru_hh_android_debug_pa
getAllExperiments() Experiment>
onAcceptExperiment(Experiment) void
ExperimentsColony
ui
R
features
feature_toggles_playground.di
ftplayground
toothpick
Resources
APK signature
ru.hh.android.debug_panel.experiments.ExperimentsCollector
@Metadata(bv = {1, 0, 3}, d1 = {"\u0000$\n\u0002\u0018\u0002\n\u0002\u0010\u0000\n\u0002\b\u0002\n\u0002\u0010!\n\u0
@ExperimentsColony
/* compiled from: ExperimentsCollector.kt */
public final class ExperimentsCollector implements ColonyFounder {
private final List<Experiment> experiments = new ArrayList();
private void found__colonist__ru_hh_android_debug_panel_experiments_ExperimentsColony() {
onAcceptExperiment(new SecondFeatureExperiment());
onAcceptExperiment(new FirstFeatureExperiment());
onAcceptExperiment(new ThirdFeatureExperiment());
onAcceptExperiment(new Experiment());
}
public void found(Class cls) {
if (cls == null) {
found__colonist__ru_hh_android_debug_panel_experiments_ExperimentsColony();
return;
}
if (cls == ExperimentsColony.class) {
found__colonist__ru_hh_android_debug_panel_experiments_ExperimentsColony();
}
}
public ExperimentsCollector() {
Colonist.settle(this);
}
public final void onAcceptExperiment(Experiment experiment) {
Intrinsics.checkNotNullParameter(experiment, "experiment");
this.experiments.add(experiment);
}
public final List<Experiment> getAllExperiments() {
return this.experiments;
Code
Smali
Вызывается в конструкторе
JADX memory usage: 0,46 GB of 4,00 GB

Используем «колонию»

```
// :debug-panel → DebugPanelViewModel.kt

fun getAllExperiments(): List<Experiment> {
    return ExperimentsCollector().getAllExperiments()
}
```

Выводы

+ Рабочее решение, хоть и альфа-версия

- Нельзя отдебажить

- Магия под капотом


Делаем
выводы

Выводы

1. Проблемы с merge-конфликтами можно решить
2. Есть много способов собрать список моделей
3. С Dagger-ом проблемы сбора списка не было бы
4. Не создавайте лишних абстракций

Спросите меня о чем-нибудь :-)

1. Проблемы feature-флагов
2. Уход от merge-конфликтов
3. Способы сбора флагов по всей кодовой базе

Ссылка на слайды