

*Sart*

**Бьем по рукам при помощи  
своего Dart-линтера**

# План доклада

# План доклада

→ Как можно реализовать свой Dart линтер?

Swift — SourceKit, SwiftLint, SourceKitten Framework

Kotlin — PSI, ktlint, Detekt

Dart — ...?

# План доклада

→ **Как можно реализовать свой Dart линтер?**

Swift — SourceKit, SwiftLint, SourceKitten Framework

Kotlin — PSI, ktlint, Detekt

Dart — ...?

→ **Как писать правила линтера и фиксы для них?**

# План доклада

## → Как можно реализовать свой Dart линтер?

Swift — SourceKit, SwiftLint, SourceKitten Framework

Kotlin — PSI, ktlint, Detekt

Dart — ...?

## → Как писать правила линтера и фиксы для них?

## → Как писать архитектурные правила?



Привет! Меня зовут **Иван**.

TeamLead и ведущий Dart & Flutter разработчик.

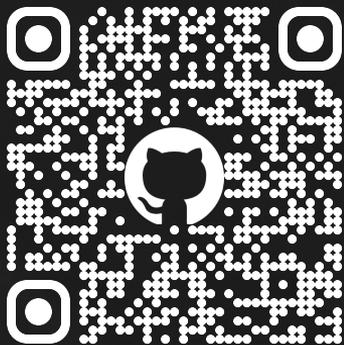
Последние 4 года занимаюсь кросс-платформенными проектами на Flutter — iOS, Android и Web.

Автор кучи полезных пакетов: `screen_corner_radius`, `pedant`, `dependency_initializer`, `flutter_platform_component`, `ellipsized_text`, `phone_number_mask_parser`, `story_opener`...

Последний год углубляюсь в Swift, UIKit, SwiftUI.

# Навигация по докладу

1) Весь приведенный код всегда будет доступен здесь:



`mobius_custom_linter_report`

2) Нумерация слайдов

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Проблемы:

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Проблемы:

- Завязываем код и архитектуру исходя из неправильных решений, принятых в начале проекта

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Проблемы:

- Завязываем код и архитектуру исходя из неправильных решений, принятых в начале проекта
- Не можем принимать какие-то решения в моменте или начинать рефакторинг из-за внешних факторов

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Проблемы:

- Завязываем код и архитектуру исходя из неправильных решений, принятых в начале проекта
- Не можем принимать какие-то решения в моменте или начинать рефакторинг из-за внешних факторов
- Доходим до точки, когда любые модификации разрушают другие части проекта

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Аспекты:

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Аспекты:

- Нужна ли вообще этому проекту стандартизация?

# Проектная деятельность

**Архитектура, масштабирование и стандартизация** — большая и больная проблема всех проектов.

Аспекты:

- Нужна ли вообще этому проекту стандартизация?
- Как реализовывать и принимать свою стандартизацию?

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Проблемы:

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Проблемы:

- Слишком долгое внедрение в проект нового разработчика

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Проблемы:

- Слишком долгое внедрение в проект нового разработчика
- Малое критическое отношение к своему коду

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Проблемы:

- Слишком долгое внедрение в проект нового разработчика
- Малое критическое отношение к своему коду
- Разработчики пишут, как им кажется, правильно, по-своему или недобросовестно

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Проблемы:

- Слишком долгое внедрение в проект нового разработчика
- Малое критическое отношение к своему коду
- Разработчики пишут, как им кажется, правильно, по-своему или недобросовестно
- Разработчик занимается модификацией или реализацией новой фичи, а потом столько же времени посвящает, чтобы не сломать существующий функционал

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Аспекты:

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Аспекты:

- Как уменьшить время на адаптацию?

# Командная деятельность

**Команда разработки** — единый организм, действующий согласовано.

Аспекты:

- Как уменьшить время на адаптацию?
- Как быстрее начать приносить пользу и прибыль проекту?

# Как можно решить?

Линтер и свои собственные строгие правила — как решение проблемы консистенции проекта, модулей, адаптации и работы команды.

# Как можно решить?

Линтер и свои собственные строгие правила — как решение проблемы консистенции проекта, модулей, адаптации и работы команды.

В линтер могут войти:

- Code-Style (наименования, аргументы, ключевые слова и т. д.)
- Архитектурные правила
- Правила по зависимостям проекта
- Правила по структуре проекта

Большинство текущих линтеров не содержат подобных правил.

# Способы реализации

# Способы реализации

- **analyzer\_plugin**

Коробочный плагин для анализатора из Dart SDK, ресурсы которого можно использовать для создания своих правил.

# Способы реализации

- **analyzer\_plugin**

Коробочный плагин для анализатора из Dart SDK, ресурсы которого можно использовать для создания своих правил.

- **custom\_lint**

**custom\_lint\_builder, \_core, \_visitor, \_generator**

Своя библиотека от компании **Invertase**, создатели пакетов **melos** и **flutterfire\_cli**.

Ведет **Remi Rousselet** - автор пакета **freezed** и стейт-менеджера **riverpod**.

Надстройка над **analyzer**, более упрощенное взаимодействие с декларациями кода.

# Анализатор

Напоминаем как работает Dart-анализатор.

# Анализатор

Напоминаем как работает Dart-анализатор.

**Dart Analysis Server** — коробочный инструмент, поставляющий аналитические ресурсы, который общается с IDE посредством прогонки JSON-сообщений с кусками деклараций кода.

# Анализатор

Напоминаем как работает Dart-анализатор.

**Dart Analysis Server** — коробочный инструмент, поставляющий аналитические ресурсы, который общается с IDE посредством прогонки JSON-сообщений с кусками деклараций кода.

Под капотом:

- Статический, лексический и семантический анализ
- Проверка типов
- Линтинг
- Детерминирование мертвого кода

# analyzer\_plugin

Первый способ, который вряд ли запустится.

# analyzer\_plugin

Первый способ, который вряд ли запустится.

Проблемы:

- Выполнение по инструкции не гарантирует старта плагина
- Функциональный подход — правила выполнены в виде синхронных генераторов
- Меньше возможностей — например на отладку, логирование и конфигурация правил

# analyzer\_plugin

Пример правила — синхронный генератор, который анализирует все классы на определение постфикса в названии.

```
1 Iterable<AnalysisErrorFixes> validateDeleteClassPostfix(  
2     String path,  
3     ResolvedUnitResult unit,  
4 ) sync* {  
5     final LibraryElement lib = unit.libraryElement;  
6     final Iterable<ClassElement> topLevelClasses =  
7         lib.topLevelElements.whereType<ClassElement>();  
8     for (final ClassElement element in topLevelClasses) {  
9         if (element.name.endsWith("Model")) {  
10            final Location location = Location(  
11                path,  
12                element.nameOffset,  
13                element.nameLength,  
14                0,  
15                0,  
16            );  
17  
18            yield AnalysisErrorFixes(  
19                AnalysisError(  
20                    AnalysisErrorSeverity.ERROR,  
21                    AnalysisErrorType.SYNTACTIC_ERROR,  
22                    location,  
23                    "Delete postfix in class",  
24                    "custom_lint_delete_class_postfix_rule",  
25                    correction: "Rename class",  
26                    hasFix: true,  
27                ),  
28            );  
29        }  
30    }  
31 }
```

# **custom\_lint, custom\_lint\_builder**

Второй способ, с которым уже можно работать.

# custom\_lint, custom\_lint\_builder

Второй способ, с которым уже можно работать.

Сложности библиотеки **custom\_lint**:

- Недостаточно примеров для реализации более сложных правил
- Приходится копаться в свойствах объектов деклараций

# custom\_lint, custom\_lint\_builder

Второй способ, с которым уже можно работать.

Сложности библиотеки **custom\_lint**:

- Недостаточно примеров для реализации более сложных правил
- Приходится копаться в свойствах объектов деклараций

Перейдем к работе.

Для начала реализуем основные ресурсы линтера.

Далее перед работой по написанию своих правил необходимо будет вспомнить основные определения для понимания работы с ними.

# Конфигурация custom\_lint

1) Реализуем **entry point**.

Для конфигурации необходимо в корневом файле пакета реализовать функцию **createPlugin**, которая возвращает экземпляр наш линтера.

```
1 // custom_lint Configuration
2 // /lib/library_name.dart
3 PluginBase createPlugin() => _ExampleLinter();
```

# Конфигурация custom\_lint

2) Создаем класс линтера.

Кастомный линтер является наследником **PluginBase**.

```
1 // custom_lint Configuration
2 // /lib/library_name.dart
3 PluginBase createPlugin() => _ExampleLinter();
4
5 class _ExampleLinter extends PluginBase {
6   _ExampleLinter();
7
8   @override
9   List<LintRule> getLintRules(
10     CustomLintConfigs configs,
11   ) =>
12     const [
13       // Перечисляем написанные правила
14       CustomLintClassDeletePostfixRule(),
15       CustomLintDeleteBlocCubitDependsFlutterRule(),
16     ];
17 }
```

# Перед началом работы Определения — Декларации

Весь код, который Вы пишете — **это декларации.**

Декларации переменных, классов, функций  
и т. д. Простым языком — утверждения,  
определяющие ресурсы программы.

TopLevelVariableDeclaration  
FunctionDeclaration  
FunctionBody  
ClassDeclaration  
ConstructorDeclaration  
VariableDeclaration

```
1 final String globalVariable = "Some string";  
2  
3 void doSome() {  
4     print(globalVariable);  
5 }  
6  
7 class SomeClass {  
8     const SomeClass({  
9         required this.intProperty,  
10        required this.stringProperty,  
11    });  
12  
13    final int intProperty;  
14    final String stringProperty;  
15  
16    void greetings() {  
17        print(intProperty);  
18        print(stringProperty);  
19    }  
20 }  
21
```

# Перед началом работы

## Определения — Элементы

**Элемент декларации** — семантическая модель программы, которая репрезентует декларации кода, названные каким-то именем, а, следовательно, на такие декларации можно ссылаться в другом месте кода.

Образуют иерархию таких абстракций для обращения и удобства.

```
1 abstract class Element implements AnalysisTarget {
2     List<Element> get children;
3
4     Element? get declaration;
5
6     String get displayName;
7
8     bool get hasAlwaysThrows;
9
10    bool get hasDeprecated;
11
12    bool get hasFactory;
13
14    bool get hasImmutable;
15
16    ...
17
18    bool get hasIsTest;
19
20    bool get hasIsTestGroup;
21
22    bool get hasLiteral;
23
24    bool get hasMustBeConst;
25 }
```

# Перед началом работы

## Определения — Токены

**Токен** — минимальная значимая единица синтаксиса, проще говоря самый минимальный элемент кода.

Ключевые слова: `if`, `else`, `return`, `yield`, `class` ...

Литералы: `47`, `"Hello"`, `[]`, `{}`, `3.14` ...

Операторы: `+`, `-`, `==`, `||`, `&&` ...

Разделители: `{ }`, `,`, `;` ...

В анализе деклараций образуют связный список, а, следовательно, можно понять кто за кем следует, а также получить конкретную позицию **токена**.

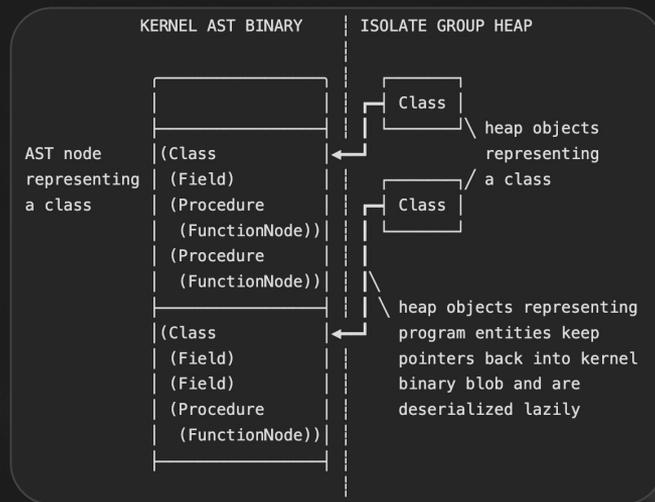
# Перед началом работы

## Определения — AST

**AST (Abstract Syntax Tree)** — конечное размеченное операторами языка ориентированное дерево, вершины которого представлены операндами.

Примечательно для **DartVM**, AST формируется на каскаде **CFE (Common Front-End)** и передается в разных режимах в виртуальную машину.

В анализе деклараций — идентичная AST.



# Шаблон правила

Любое правило — наследник **DartRuleLint**.

Для работы и анализа правила необходимо указать объект кода правила и реализовать метод **run()**. Дополнительно указываем на фиксы.

```
1 class CustomLintTemplateRule extends DartLintRule {
2   const CustomLintTemplateRule()
3     : super(
4       code: const LintCode(
5         name: 'custom_lint_template_rule',
6         problemMessage: 'This is the description of rule',
7         errorSeverity: error.ErrorSeverity.ERROR,
8       ),
9     );
10
11   @override
12   void run(
13     CustomLintResolver resolver,
14     ErrorReporter reporter,
15     CustomLintContext context,
16   ) {
17     // Логика правила
18   }
19
20   // Список фиксов, предоставляемое для правила
21   @override
22   List<Fix> getFixes() => [
23     _Fix(),
24   ];
25 }
26
27 class _Fix extends DartFix {
28   _Fix();
29
30   @override
31   void run(
32     CustomLintResolver resolver,
33     ChangeReporter reporter,
34     CustomLintContext context,
35     error.AnalysisError analysisError,
36     List<error.AnalysisError> others,
37   ) {
38     // Логика фикса
39   }
40 }
```

# Реализуем простое правило Постфикс в названии класса

1) Делаем первое простое правило с декларацией класса и его постфиксом в названии.

```
1 class CustomLintClassDeletePostfixRule extends DartLintRule {
2   static const List<String> deletePostfixes = [
3     "Impl",
4     "Implementation",
5     "Model",
6   ];
7
8   const CustomLintClassDeletePostfixRule()
9     : super(
10      code: const LintCode(
11        name: 'custom_lint_delete_class_postfix_rule',
12        problemMessage: 'This is the description of rule',
13        errorSeverity: error.ErrorSeverity.ERROR,
14      ),
15    );
16
17   @override
18   void run(
19     CustomLintResolver resolver,
20     ErrorReporter reporter,
21     CustomLintContext context,
22   ) {}
23 }
```

# Реализуем простое правило Постфикс в названии класса

2) Реализуем логику проверки деклараций классов.

```
1  @override
2  void run(
3    CustomLintResolver resolver,
4    ErrorReporter reporter,
5    CustomLintContext context,
6  ) =>
7    context.registry.addClassDeclaration(
8      (
9        ClassDeclaration classDeclaration,
10     ) {
11       ...
12     },
13   );
```

# Реализуем простое правило Постфикс в названии класса

2.1) Проверка наличие элемента декларации.

```
1 final ClassElement? classElement = classDeclaration.declaredElement;  
2 if (classElement == null) {  
3     return;  
4 }
```

# Реализуем простое правило Постфикс в названии класса

2.2) Проверка совпадения в имени элемента.

```
1  for (final String postfix in deletePostfixes) {  
2    if (classElement.displayName.endsWith(  
3      postfix,  
4      ) ==  
5      false) {  
6      continue;  
7    }  
8  
9    ...  
10 }
```

# Реализуем простое правило Постфикс в названии класса

2.3) Передаем в **reporter** о наличии ошибки в элементе.

```
1  for (final String postfix in deletePostfixes) {
2    if (classElement.displayName.endsWith(
3        postfix,
4        ) ==
5        false) {
6        continue;
7    }
8
9    return reporter.atElement(
10       classElement,
11       code,
12     );
13 }
```

# Реализуем простое правило Постфикс в названии класса

2) Реализуем логику проверки деклараций классов.

2.1) Проверка наличие элемента декларации.

2.2) Проверка совпадения в имени элемента.

2.3) Передаем в **reporter** о наличии ошибки в элементе.

```
1 @Override
2 void run(
3     CustomLintResolver resolver,
4     ErrorReporter reporter,
5     CustomLintContext context,
6 ) =>
7     context.registry.addClassDeclaration(
8         (
9             ClassDeclaration classDeclaration,
10            ) {
11                final ClassElement? classElement = classDeclaration.declaredElement;
12                if (classElement == null) {
13                    return;
14                }
15
16                for (final String postfix in deletePostfixes) {
17                    if (classElement.displayName.endsWith(
18                        postfix,
19                    ) =
20                        false) {
21                        continue;
22                    }
23
24                    return reporter.atElement(
25                        classElement,
26                        code,
27                    );
28                }
29            },
30 );
```

# Реализуем простое правило Постфикс в названии класса

3) Проверка работы.

```
custom_lint_delete_class_postfix_rule.dart ×  
example > lib > custom_lint_delete_class_postfix_rule.dart > SomeClass  
1 class SomeClass {}  
2
```

# Реализуем фикс правила

Реализуем для готового правила фикс, который можно исправить вручную или через CLI.

1) Реализуем в заготовленном правиле метод предоставления списка фиксов.

```
1 @override
2 List<Fix> getFixes() => [
3     _Fix(),
4 ];
```

# Реализуем фикс правила

2) Реализуем шаблон фикса.

Любой фикс — наследник **DartFix**.

Для работы фикса кода необходимо также реализовать метод **run()**, и внутри создаем **ChangeBuilder**.

```
1 class _Fix extends DartFix {
2   _Fix();
3
4   @override
5   void run(
6     CustomLintResolver resolver,
7     ChangeReporter reporter,
8     CustomLintContext context,
9     error.AnalysisError analysisError,
10    List<error.AnalysisError> others,
11   ) {}
12 }
13
```

# Реализуем фикс правила

3) Реализуем логику фикса.

```
1  @override
2  void run(
3      CustomLintResolver resolver,
4      ChangeReporter reporter,
5      CustomLintContext context,
6      error.AnalysisError analysisError,
7      List<error.AnalysisError> others,
8  ) =>
9      context.registry.addClassDeclaration(
10         (
11             ClassDeclaration classDeclaration,
12         ) {
13             ...
14         },
15     );
```

# Реализуем фикс правила

3.1) Проверка на пересечение полученной ошибки.

```
1  if (analysisError.sourceRange.intersects(  
2      classDeclaration.sourceRange,  
3      ) ==  
4      false) {  
5      return;  
6  }
```

# Реализуем фикс правила

3.2) Повторная проверка на наличия элемента декларации.

```
1 final ClassElement? classElement = classDeclaration.declaredElement;  
2 if (classElement == null) {  
3     return;  
4 }
```

# Реализуем фикс правила

3.3) Определяем валидное имя элемента.

```
1 String validName = classElement.displayName;
2 for (final String postfix
3     in CustomLintClassDeletePostfixRule.deletePostfixes) {
4     if (classElement.displayName.endsWith(
5         postfix,
6         ) ==
7         false) {
8         continue;
9     }
10
11     validName = validName.substring(
12         0,
13         validName.length - postfix.length,
14     );
15 }
```

# Реализуем фикс правила

3.4) Создаем **ChangeBuilder** с логикой замены участка кода.

```
1  final ChangeBuilder changeBuilder = reporter.createChangeBuilder(  
2    message: "Fix: Rename to '$validName'",  
3    priority: 100,  
4  );  
5  changeBuilder.addDartFileEdit(  
6    (  
7      DartFileEditBuilder builder,  
8    ) =>  
9      builder.addSimpleReplacement(  
10     analysisError.sourceRange,  
11     validName,  
12   ),  
13 );
```

# Реализуем фикс правила

3) Реализуем логику фикса.

3.1) Проверка на пересечение полученной ошибки.

3.2) Повторная проверка на наличие элемента декларации.

3.3) Определяем валидное имя элемента.

3.4) Создаем **ChangeBuilder** с логикой замены участка кода.

```
1 @override
2 void run(
3   CustomLintResolver resolver,
4   ChangeReporter reporter,
5   CustomLintContext context,
6   error.AnalysisError analysisError,
7   List<error.AnalysisError> others,
8 ) =>
9   context.registry.addClassDeclaration(
10    {
11      ClassDeclaration classDeclaration,
12    } {
13      if (analysisError.sourceRange.intersects(
14        classDeclaration.sourceRange,
15      ) ==
16        false) {
17        return;
18      }
19
20      final ClassElement? classElement = classDeclaration.declaredElement;
21      if (classElement == null) {
22        return;
23      }
24
25      String validName = classElement.displayName;
26      for (final String postfix
27        in CustomLintClassDeletePostfixRule.deletePostfixes) {
28        if (classElement.displayName.endsWith(
29          postfix,
30        ) ==
31          false) {
32          continue;
33        }
34
35        validName = validName.substring(
36          0,
37          validName.length - postfix.length,
38        );
39      }
40
41      final ChangeBuilder changeBuilder = reporter.createChangeBuilder(
42        message: "Fix: Rename to '$validName'",
43        priority: 100,
44      );
45      changeBuilder.addDartFileEdit(
46        (
47          DartFileEditBuilder builder,
48        ) =>
49          builder.addSimpleReplacement(
50            analysisError.sourceRange,
51            validName,
52          ),
53      );
54    },
55  );
```

# Реализуем фикс правила

4) Проверка работы.

```
custom_lint_delete_class_postfix_rule.dart ×  
example > lib > custom_lint_delete_class_postfix_rule.dart > ...  
1  class SomeClass {}  
2  
   I
```

# Архитектурное правило

Двигаемся дальше — реализуем архитектурное правило.

Возьмем популярный стейт-менеджмент **BLoC** и попробуем ограничить его зависимости.

Любой блок не сможет быть зависим от ресурсов **Flutter SDK**.

Реализуем **TypeChecker** для проверки типа декларации класса Bloc и Cubit, имплементируем правило и сразу фикс для него.

# Архитектурное правило

1) Заготавливаем объекты проверки типа класса — **TypeChecker**.

```
1  const TypeChecker blocTypeChecker = TypeChecker.fromName(  
2    "Bloc",  
3    packageName: "bloc",  
4  );  
5  const TypeChecker cubitTypeChecker = TypeChecker.fromName(  
6    "Cubit",  
7    packageName: "bloc",  
8  );
```

# Архитектурное правило

2) Реализуем логику правила.

Запускаем анализ деклараций классов и проверяем на соответствие их типов.

```
1 context.registry.addClassDeclaration(  
2   (  
3     ClassDeclaration classDeclaration,  
4   ) {  
5     final ClassElement? classElement = classDeclaration.declaredElement;  
6     if (classElement == null) {  
7       return;  
8     }  
9  
10    if (!blocTypeChecker.isAssignableFrom(  
11      classElement,  
12    ) &&  
13      !cubitTypeChecker.isAssignableFrom(  
14        classElement,  
15      )) {  
16        return;  
17      }  
18  
19      _visitChildren(  
20        reporter: reporter,  
21        classDeclaration: classDeclaration,  
22      );  
23    },  
24  );
```

# Архитектурное правило

3) В отдельном методе при помощи визитера проверяем конструктор, блок инициализации этого конструктора и поля самого класса.

```
1 void _visitChildren({
2     required ErrorReporter reporter,
3     required ClassDeclaration classDeclaration,
4 }) =>
5     classDeclaration.visitChildren(
6         AstTreeVisitor(
7             onConstructorDeclaration: (
8                 ConstructorDeclaration constructorDeclaration,
9             ) {
10                ...
11            },
12            constructorDeclaration.visitChildren(
13                AstTreeVisitor(
14                    onConstructorFieldInitializer: (
15                        ConstructorFieldInitializer constructorFieldInitializer,
16                    ) {
17                        ...
18                    },
19                ),
20            );
21        },
22        onFieldDeclaration: (
23            FieldDeclaration fieldDeclaration,
24        ) {
25            ...
26        }
27    ),
28 );
29 );
```

# Архитектурное правило

## 3.1) Проверка декларации конструктора.

```
1  onConstructorDeclaration: (  
2    ConstructorDeclaration constructorDeclaration,  
3  ) {  
4    for (final FormalParameter parameter  
5      in constructorDeclaration.parameters.parameters) {  
6      final ParameterElement? parameterElement =  
7        parameter.declaredElement;  
8      if (parameterElement == null) {  
9        continue;  
10     }  
11  
12     if (!_validate(  
13       packageName:  
14         parameterElement.type.element?.library?.identifier,  
15       ) ==  
16       false) {  
17       continue;  
18     }  
19  
20     reporter.atElement(  
21       parameterElement,  
22       code,  
23     );  
24   }
```

# Архитектурное правило

3.2) Проверка декларации полей в блоке инициализации конструктора.

```
1 onConstructorFieldInitializer: (  
2   ConstructorFieldInitializer constructorFieldInitializer,  
3 ) {  
4   final DartType? staticType =  
5     constructorFieldInitializer.expression.staticType;  
6   if (staticType == null) {  
7     return;  
8   }  
9  
10  if (_validate(  
11    packageName: staticType.element?.library?.identifier,  
12    ) ==  
13    false) {  
14    return;  
15  }  
16  
17  reporter.atNode(  
18    constructorFieldInitializer,  
19    code,  
20  );  
21 },
```

# Архитектурное правило

## 3.3) Проверка полей класса.

```
1 onFieldDeclaration: (  
2   FieldDeclaration fieldDeclaration,  
3 ) {  
4   for (final VariableDeclaration variable  
5     in fieldDeclaration.fields.variables) {  
6     final VariableElement? variableElement = variable.declaredElement;  
7     if (variableElement == null) {  
8       continue;  
9     }  
10  
11     if (!_validate(  
12       packageName:  
13         variableElement.type.element?.library?.identifier,  
14       ) ==  
15         false) {  
16       continue;  
17     }  
18  
19     reporter.atNode(  
20       fieldDeclaration,  
21       code,  
22     );  
23   }  
24 },
```

# Архитектурное правило

4) Отдельно выносим функцию, в которой и валидируем имя пакета анализируемого ресурса.

```
1  bool _validate({
2    required String? packageName,
3  }) =>
4    packageName?.startsWith(
5      "package:flutter/",
6    ) ??
7    false;
```

# Архитектурное правило

5) Основная логика фикса — удаляем аргументы конструктора.

```
1 final ChangeBuilder changeBuilder = reporter.createChangeBuilder(  
2     message: "Fix: Delete '${parameterElement.displayName}'",  
3     priority: 100,  
4 );  
5 changeBuilder.addDartFileEdit(  
6     (  
7         DartFileEditBuilder builder,  
8     ) =>  
9         builder.addDeletion(  
10            SourceRange(  
11                formalParameter.sourceRange.offset,  
12                formalParameter.endToken.next?.type == TokenType.COMMA  
13                    ? (formalParameter.sourceRange.length + 1)  
14                    : formalParameter.sourceRange.length,  
15            ),  
16        ),  
17    );
```

# Архитектурное правило

5) Основная логика фикса — удаляем поля инициализации конструктора.

```
1  final ChangeBuilder changeBuilder = reporter.createChangeBuilder(  
2    message: "Fix: Delete '${constructorFieldInitializer.fieldName}'",  
3    priority: 100,  
4  );  
5  changeBuilder.addDartFileEdit(  
6    (  
7      DartFileEditBuilder builder,  
8    ) =>  
9      builder.addDeletion(  
10     SourceRange(  
11       constructorFieldInitializer.sourceRange.offset,  
12       constructorFieldInitializer.endToken.next?.type == TokenType.COMMA  
13         ? (constructorFieldInitializer.sourceRange.length + 1)  
14         : constructorFieldInitializer.sourceRange.length,  
15     ),  
16   ),  
17 );
```

# Архитектурное правило

5) Основная логика фикса — удаляем поля (переменные) класса.

```
1  final ChangeBuilder changeBuilder = reporter.createChangeBuilder(  
2    message: "Fix: Delete '${variableElement.displayName}'",  
3    priority: 100,  
4  );  
5  changeBuilder.addDartFileEdit(  
6    (  
7      DartFileEditBuilder builder,  
8    ) =>  
9      builder.addDeletion(  
10     analysisError.sourceRange,  
11   ),  
12 );
```

# Архитектурное правило

6) Проверка работы.

```
custom_lint_delete_bloc_cubit_depends_flutter.dart 9 x
example > lib > custom_lint_delete_bloc_cubit_depends_flutter.dart > Example1Bloc > Example1Bloc
1  import 'package:flutter/material.dart';
2
3  import 'package:bloc/bloc.dart';
4
5  class Example1Bloc extends Bloc<String, String> {
6    Example1Bloc(
7      this._animationController, {
8      required TextEditingController textEditingController,
9      required this.scrollController,
10     } : _textEditingController = textEditingController,
11       super()
12     );
13
14
15     final AnimationController _animationController;
16     final TextEditingController _textEditingController;
17     final ScrollController scrollController;
18   }
19
```

# AST Визитеры

**Визитер** — известный паттерн проектирования.

**AstVisitor<R>** — базовый класс, реализующий коллбеки по мере движения по декларациям кода.

# AST Визитеры

**Визитер** — известный паттерн проектирования.

**AstVisitor<R>** — базовый класс, реализующий коллбеки по мере движения по декларациям кода.

На его основе имеются уже готовые подклассы:

- GeneralizingAstVisitor<R>
- RecursiveAstVisitor<R>
- SimpleAstVisitor<R>
- TimeAstVisitor<R>
- UnifyingAstVisitor<R>

# AST Визитеры

Пример своего визитера на любые случаи жизни:

```
1 class AstTreeVisitor extends RecursiveAstVisitor<void> {
2     const AstTreeVisitor({
3         this.onClassDeclaration,
4         ...
5     });
6
7     final void Function(
8         ClassDeclaration node,
9     )? onClassDeclaration;
10
11     ...
12
13     @override
14     void visitClassDeclaration(
15         ClassDeclaration node,
16     ) {
17         super.visitClassDeclaration(
18             node,
19         );
20         onClassDeclaration?.call(
21             node,
22         );
23     }
24
25     ...
26 }
```

# Отладка

custom\_lint.log

Включение отладки в  
подключенном проекте:

- 1 custom\_lint:
- 2 debug: true
- 3 verbose: true

```
[custom_lint] 2025-01-06T15:41:18.427322 /Users/ivangalikin/Development/MyA20/projects/dart/flutter/mobius_custom_lint_demo/lib
2 [custom_lint] 2025-01-06T15:41:18.427322 - "_ExampleLint": is from "package:mobius_custom_lint_demo/mobius_custom_lint_demo
3 [custom_lint] 2025-01-06T15:41:18.427322 Try correcting the name to the name of an existing method, or defining a method nam
4 [custom_lint] 2025-01-06T15:41:18.427322 MyCustomLintCode(),
5 [custom_lint] 2025-01-06T15:41:18.427322 ~~~~~
6 [custom_lint] 2025-01-06T15:41:18.427322
7 [custom_lint] 2025-01-06T15:41:18.427322
8 failed to start pluginsThe request analysis.setContextRoots failed with the following error:
9 RequestError{Code:PLUGIN_ERROR
10 Bad state: Failed to start the plugins.
11 at:
12 #0 SocketCustomLintServerToClientChannel._checkInitializationFail.<anonymous closure> (package:custom_lint/src/v2/server
13 <asynchronous suspension>
14 #1 Future.any.maybeValue (dart:async/future.dart:134:5)
15 <asynchronous suspension>
16
17 The Dart VM service is listening on http://127.0.0.1:55229/?vT3d460w/
18 The Dart DevTools debugger and profiler is available at: http://127.0.0.1:55229/?vT3d460w/devtools/?url=ws://127.0.0.1:552
19 The Dart DevTools debugger and profiler is available at: http://127.0.0.1:55305/?vR80K4c5S0@devtools/?url=ws://127.0.0.1:553
20 The Dart VM service is listening on http://127.0.0.1:55633/?vG6tbe-Vlmgw/
21 The Dart DevTools debugger and profiler is available at: http://127.0.0.1:55633/?vG6tbe-Vlmgw-Http?devtools/?url=ws://127.0.0.1:556
22 The Dart VM service is listening on http://127.0.0.1:55676/?vR2c3-Cd4d
23 The Dart DevTools debugger and profiler is available at: http://127.0.0.1:55676/?vR2c3-Cd4d@devtools/?url=ws://127.0.0.1:556
24 [custom_lint] 2025-01-06T15:02:23.443031 /Users/ivangalikin/Development/MyA20/projects/dart/flutter/mobius_custom_lint_demo/lib
25 [custom_lint] 2025-01-06T15:02:23.443031 export "err/custom_lint_package/custom_lint_delete_class_postfix_rule.dart";
26 [custom_lint] 2025-01-06T15:02:23.443031
27 [custom_lint] 2025-01-06T15:02:23.443031 /Users/ivangalikin/Development/MyA20/projects/dart/flutter/mobius_custom_lint_demo/lib
28 [custom_lint] 2025-01-06T15:02:23.443031 CustomLintClassDeletePostfixRule(),
29 [custom_lint] 2025-01-06T15:02:23.443031 ~~~~~
30 [custom_lint] 2025-01-06T15:02:23.443031
31 [custom_lint] 2025-01-06T15:02:23.443031
32 failed to start pluginsThe request analysis.setContextRoots failed with the following error:
33 RequestError{Code:PLUGIN_ERROR
34 Bad state: Failed to start the plugins.
35 at:
36 #0 SocketCustomLintServerToClientChannel._checkInitializationFail.<anonymous closure> (package:custom_lint/src/v2/server
37 <asynchronous suspension>
38 #1 Future.any.maybeValue (dart:async/future.dart:134:5)
39 <asynchronous suspension>
40
41 [custom_lint] 2025-01-06T15:02:23.443051 _PrettyRequestFailure: {"code":"PLUGIN_ERROR","message":"Bad state: Failed to start
42 [custom_lint] 2025-01-06T15:02:23.443051 # ChannelBase.sendRequest (package:custom_lint/src/server_isolate_channel.dart
43 [custom_lint] 2025-01-06T15:02:23.443051 <asynchronous suspension>
44 [custom_lint] 2025-01-06T15:02:23.443051 # CustomLintRunner.initialize.<anonymous closure> (package:custom_lint/src/run
45 [custom_lint] 2025-01-06T15:02:23.443051 <asynchronous suspension>
46 [custom_lint] 2025-01-06T15:02:23.443051 # _runServer.<anonymous closure> (package:custom_lint/custom_lint.dart:90:7)
47 [custom_lint] 2025-01-06T15:02:23.443051 <asynchronous suspension>
48 [custom_lint] 2025-01-06T15:02:23.443051 # _runServer.<anonymous closure> (package:custom_lint/custom_lint.dart:126:13)
49 [custom_lint] 2025-01-06T15:02:23.443051 # _runServer.<anonymous closure> (package:custom_lint/custom_lint.dart:126:13)
50 [custom_lint] 2025-01-06T15:02:23.443051 # _runServer.<anonymous closure> (package:custom_lint/custom_lint.dart:126:13)
51 [custom_lint] 2025-01-06T15:02:23.443051 # customLint (package:custom_lint/custom_lint.dart:150:5)
52 [custom_lint] 2025-01-06T15:02:23.443051 <asynchronous suspension>
53 [custom_lint] 2025-01-06T15:02:23.443051 # entrypoint (file:///Users/ivangalikin/.pub-cache/hosted/pub.dev/custom_lint-4
54 [custom_lint] 2025-01-06T15:02:23.443051 <asynchronous suspension>
55 [custom_lint] 2025-01-06T15:02:23.443051 # main (file:///Users/ivangalikin/.pub-cache/hosted/pub.dev/custom_lint-4.7.0:
56 [custom_lint] 2025-01-06T15:02:23.443051 <asynchronous suspension>
57 [custom_lint] 2025-01-06T15:02:23.443051
58 [custom_lint] 2025-01-06T15:02:23.443051 /Users/ivangalikin/Development/MyA20/projects/dart/flutter/mobius_custom_lint_demo/lib
59 [custom_lint] 2025-01-06T15:02:23.443051 export "err/custom_lint_package/custom_lint_delete_class_postfix_rule.dart";
60 [custom_lint] 2025-01-06T15:02:23.443051
61 [custom_lint] 2025-01-06T15:02:23.443051 /Users/ivangalikin/Development/MyA20/projects/dart/flutter/mobius_custom_lint_demo/lib
62 [custom_lint] 2025-01-06T15:02:23.443051 CustomLintClassDeletePostfixRule(),
63 [custom_lint] 2025-01-06T15:02:23.443051 ~~~~~
64 [custom_lint] 2025-01-06T15:02:23.443051
65 [custom_lint] 2025-01-06T15:02:23.443051
```



# Тесты

Тест написанных правил:

```
1 // expect_lint: your_linter_rule
```

custom\_lint\_delete\_class\_postfix\_rule.dart ×

example > lib > custom\_lint\_delete\_class\_postfix\_rule.dart > ...

```
1 // expect_lint: custom_lint_delete_class_postfix_rule
2 class SomeClassModel {}
3 |
```

# Тесты

**Писать правила — не сложно.**

То, что может без проблем написать каждый:

# Тесты

**Писать правила — не сложно.**

То, что может без проблем написать каждый:

- Постфиксы и префиксы классов по типу базового класса или расположения этого файла

# Тесты

**Писать правила — не сложно.**

То, что может без проблем написать каждый:

- Постфиксы и префиксы классов по типу базового класса или расположения этого файла
- Визуальный стиль — запятые, скобки, типы переменных или виды аргументов

# Тесты

**Писать правила — не сложно.**

То, что может без проблем написать каждый:

- Постфиксы и префиксы классов по типу базового класса или расположения этого файла
- Визуальный стиль — запятые, скобки, типы переменных или виды аргументов
- Удаление nereкомендуемых ресурсов — пакеты, типы (Container, Eather и т. д.) или функции (print, функции возвращающие Widget и т. д.)

# Тесты

## Писать правила — не сложно.

То, что может без проблем написать каждый:

- Постфиксы и префиксы классов по типу базового класса или расположения этого файла
- Визуальный стиль — запятые, скобки, типы переменных или виды аргументов
- Удаление нерекондуемых ресурсов — пакеты, типы (Container, Eather и т. д.) или функции (print, функции возвращающие Widget и т. д.)
- Наименования файлов и их расположение в структуре проекта

# Нюансы — Ключевые слова

Трудно реализовывать правила с ключевыми словами, поскольку для реализации таких правил все больше необходим контекст их применения в синтаксисе.

# Нюансы — Ключевые слова

Трудно реализовывать правила с ключевыми словами, поскольку для реализации таких правил все больше необходим контекст их применения в синтаксисе.

Пример обсуждения правил, связанных с ключевым словом **const**:

Compiler should try to make everything const #1410

🔒 Closed

Investigate constness #149932

🔒 Closed

[proposal] add const at compile time (where possible) #56669

🔒 Closed

# Нюансы — Совместимость

Совместимость с другими пакетами в контексте **include**-директивы.

# Нюансы — Совместимость

Совместимость с другими пакетами в контексте **include**-директивы.

## **flutter\_lints**

include: package:flutter\_lints/flutter.yaml

## **lint**

include: package:lint/strict.yaml -/casual.yaml -/package.yaml

## **lints**

include: package:lints/recommended.yaml -/core.yaml

## **very\_good\_analysis**

include: package:very\_good\_analysis/analysis\_options.yaml

## **surf\_lint\_rules**

include: package:surf\_lint\_rules/analysis\_options.yaml

# Нюансы — Анализ

Анализ происходит только в рамках текущего сканируемого файла.

Не получится репортировать об ошибках со связанными ресурсами в соседних файлах.

# Нюансы — Анализ

Анализ происходит только в рамках текущего сканируемого файла.

Не получится репортировать об ошибках со связанными ресурсами в соседних файлах.

Пример — определять, соответствует ли класс State у Bloc нашим требованиям по декларации в текущем файле, а указывать об ошибке этого класса в другом файле.

# Нюансы — Комбинирование

Комбинирование правил и конфигурация по параметрам.

Напоминаю, что можно в **analysis\_options.yaml** перечислять включение/выключение правил и передачу в них параметров.

```
1  custom_lint:  
2    enable_all_lint_rules: false  
3    rules:  
4      - my_lint_rule_0: false  
5      - my_lint_rule_1:  
6        some_parameter: "some value"
```

# CLI & UseCases

## CLI:

Где использовать:

- Скрипты уборки проекта
- GitHooks (pre-commit)

Анализ и запуск фиксов: `dart run custom_lint -watch / -fix`

# CLI & UseCases

## CLI:

Где использовать:

- Скрипты уборки проекта
- GitHooks (pre-commit)

Анализ и запуск фиксов: `dart run custom_lint -watch / -fix`

## CI/CD?

Необходимость в уборке проекта на этом этапе минимальная, поскольку в репозиторий на этапе MR/PR уже должна вливаться подготовленная и убранная кодовая база.

# Профит

Итоговый результат:

# Профит

Итоговый результат:

- Тотальная консистенция всей кодовой базы, нет разницы между стилем и подходами в коде в любом месте проекта

# Профит

Итоговый результат:

- Тотальная консистенция всей кодовой базы, нет разницы между стилем и подходами в коде в любом месте проекта
- Минимизация ошибок при разработке, как архитектурные так и из-за человеческого фактора

# Профит

Итоговый результат:

- Тотальная консистенция всей кодовой базы, нет разницы между стилем и подходами в коде в любом месте проекта
- Минимизация ошибок при разработке, как архитектурные так и из-за человеческого фактора
- Быстрое внедрение нового разработчика, который видит ограничения не только из документации к проекту, но также из правил при непосредственной работе с проектом

# Опыт

Реализовал несколько насущных архитектурных правил, а также черный список — [github.com/ivangalkindeveloper/pedant](https://github.com/ivangalkindeveloper/pedant).



**pedant** Public ☆ Star

Static strict analyzer and linter for Dart & Flutter, contains script for fixing errors, convert declarations and formatting code

[dart](#) [dartlang](#) [dart-library](#) [dart-lang](#) [dart-package](#) [dartanalyzer](#) [dart-analyzer](#)

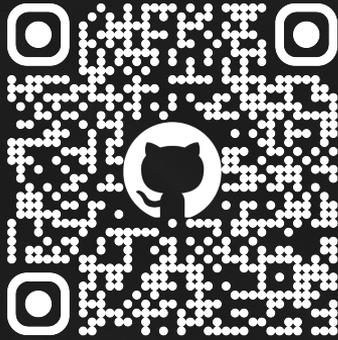
● Dart ☆ 2 🔗 Other Updated 5 hours ago

# Внимание

Спасибо за внимание ❤️

# Внимание

Спасибо за внимание ❤️



mobius\_custom\_linter\_report



Surf Tech



Surf Flutter Team