

ruleguard workshop

GopherCon Russia 2021

Какой у нас план?

- учимся запускать и настраивать ruleguard
- разбираемся с написанием новых правил
- отлаживаем их, пишем для них тесты
- обсуждаем проблемы и их решения

github.com/quasilyte/gophercon2021-ruleguard

Репозиторий проекта со всеми материалами





Старый талисман проекта



Новый талисман проекта

Что такое ruleguard?

Что такое ruleguard?

- Инструмент для рефакторинга

Что такое ruleguard?

- Инструмент для рефакторинга
- Инструмент для структурного поиска кода

Что такое ruleguard?

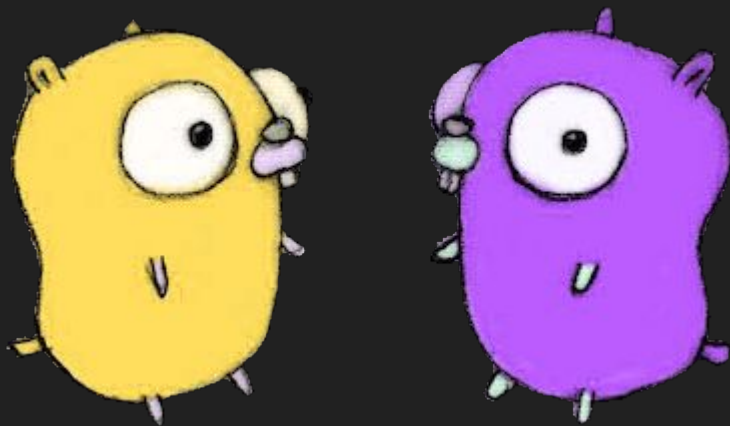
- Инструмент для рефакторинга
- Инструмент для структурного поиска кода
- **Расширяемый** статический анализатор

Что такое ruleguard?

- Инструмент для рефакторинга
- Инструмент для структурного поиска кода
- **Расширяемый** статический анализатор
- Написан на Go, для Go

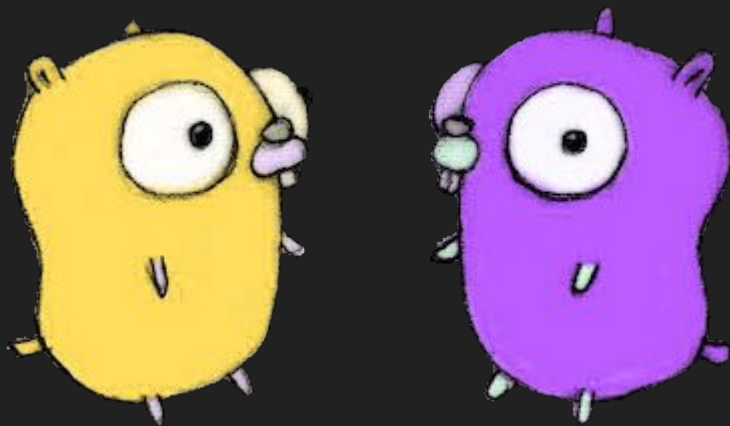
Концепция рулгарда

Из разговора с одной из конференций



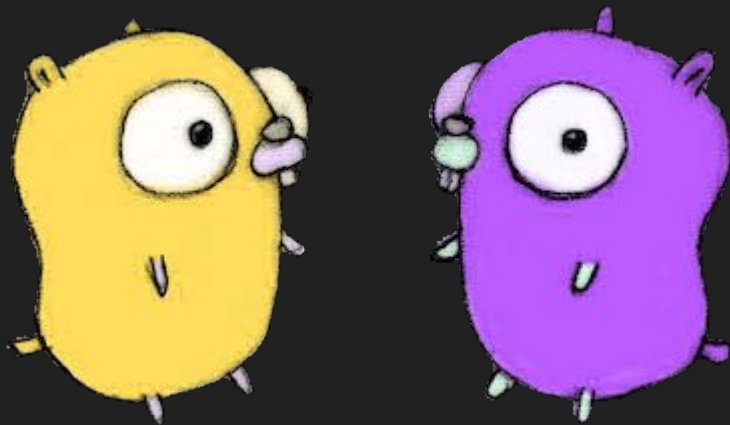
Из разговора с одной из конференций

gogrep?



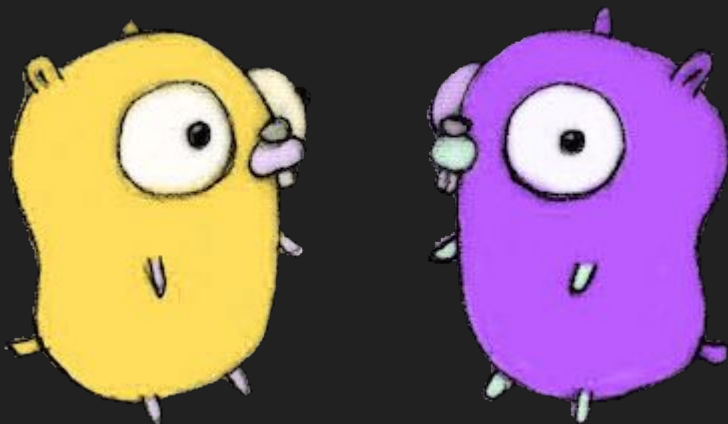
Из разговора с одной из конференций

static analysis?



Из разговора с одной из конференций

gogrep-based static
analysis!





Концепция

A special Go-compatible format

Not just gogrep anymore



Реализация, эволюция

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич
- Рекомендация лучших альтернатив

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич
- Рекомендация лучших альтернатив
- Прототипирование диагностик

Для чего он подходит лучше всего?

- Проверка стиля кода вашего проекта
- Запрет нежелательных функций/фич
- Рекомендация лучших альтернатив
- Прототипирование диагностик

Вы *сами* делаете этот линтер.

Он создан для того, чтобы его расширяли.

Сравнение с традиционными линтерами

Динамическое расширение

Рулгард расширяется динамически. После добавления новых диагностик не нужна перекомпиляция.

Декларативное описание диагностик

Вы описываете шаблоны, фильтры и сообщения для предупреждений. Ничего лишнего.

Остальное за вас делает рулгард.

Авто-интеграция в golangci-lint

Ваши правила автоматически становятся запускаемыми
через все любимый golangci-lint.

Ближайшие аналоги:

CodeQL и Semgrep

Ближайшие аналоги:


[CodeQL] и Semgrep



Высокий порог входа для написания правил.
Правила пишем на своеобразном специализированном языке
программирования.

Ближайшие аналоги:

CodeQL и **[Semgrep]**



Значительно хуже поддерживает Go.
Правила приходится “программировать” в YAML.

А у нас в ruleguard:

* Правила на Go *

* Установка правил через Go модули *

* Отличная поддержка Go *

Курс молодого бойца

Особо важные пакеты из stdlib

Особо важные пакеты из stdlib

- `go/ast` - чтобы понимать AST матчинг

Особо важные пакеты из stdlib

- `go/ast` - чтобы понимать AST матчинг
- `go/types` - для понимания фильтров

Особо важные пакеты из stdlib

- `go/ast` - чтобы понимать AST матчинг
- `go/types` - для понимания фильтров

В общем-то и всё.

Остальное инкапсулирует рулгард.

Но есть нюанс, который стоит повторить

[go/ast] vs go/types

Типы **go/ast** указывают категорию синтаксического элемента (например, “цикл”).

[go/ast] vs [go/types]

Типы **go/types** указывают на тип выражения или декларации, которое выводится не только исходя из синтаксической структуры.

`int(15)`

AST type = `CallExpr`

Expression type = `int`

```
Node.Is("CallExpr") && Type.Is("int")
```

Пример фильтра, демонстрирующий важность этого
разделения

Справку по терминологии ищи в
“`notes/terminology.md`”
внутри репозитория

А теперь приступим!