Санкт-Петербургский государственный университет
Кафедра системного программирования

Ускорение фильтрации трафика антибот-системы с использованием библиотеки исполнения выражений

Григорий Александрович Асеев, группа 21.Б15-мм

Научный руководитель: Д. В. Луцив, доцент кафедры системного программирования, к.ф.-м.н.
Консультант: И. А. Мерочкин, разработчик информационных систем ООО «Озон Технологии»

Рецензент: Е. М. Гришанков, программист ООО «Озон Технологии»

Санкт-Петербург
2025

Введение

- Фильтрация трафика
- Антибот-система Ozon
  - Ручной рейт-лимитер
  - Ежедневно изменяемые правила
    - `ja4TLS not in JA4TLSBot && rate("1s", EntityIP) > 600 && runModel("tlsUAMobile")`
- Библиотека исполнения правил на основе замыканий

Постановка задачи

Цель: разработать библиотеку для высокопроизводительной фильтрации трафика в антибот-системах, обеспечивающей совместимость с текущими правилами антибота Ozon.

Задачи:

- Описать процесс фильтрации трафика с использованием правил в антибот-системе
- Провести анализ существующих библиотек и подходов для исполнения выражений
- Спроектировать библиотеку для высокопроизводительной фильтрации трафика в антибот-системах
- Выполнить реализацию указанной библиотеки, включающую:
  - Парсер выражений
  - Компилятор преобразующий выражения в замыкания
  - Механизм доступа к переменным
  - Обеспечение полной совместимости синтаксиса с текущими правилами антибот-системы
- Разработать набор тестов для библиотеки, интегрировать её в сервисы и провести сравнительный анализ производительности с существующими решениями.

Фильтрация трафика с использованием правил

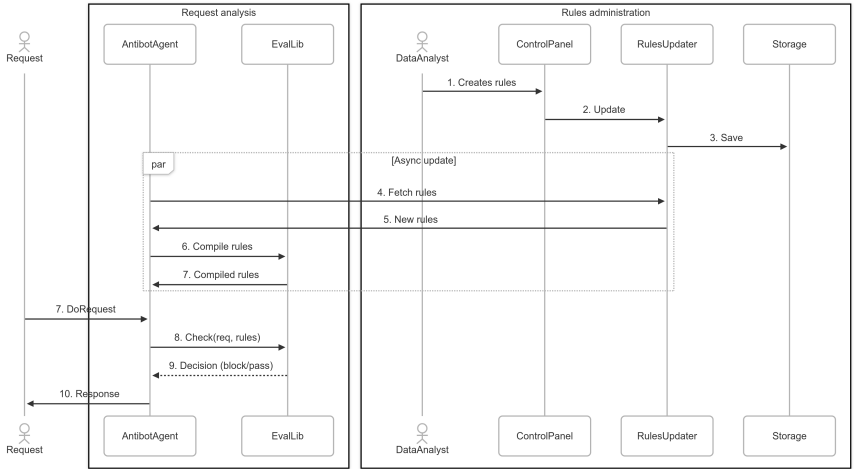

Рис. 1: Диаграмма последовательностей фильтрации трафика

Существующие библиотеки исполнения выражений

- Govaluate
  - Интерпретирует выражения без предварительных оптимизаций
  - Работает с переменными обобщенного типа, требует уже готовые значения
- Expr
  - Предварительные оптимизации: константное свертывание, вызов функций с известными аргументами
  - Стековая виртуальная машина
    - ★ Можно переиспользовать для последующих вызовов
  - Работа с переменными аналогична Govaluate

Подход на основе замыканий

- Представление каждого узла выражения в виде функции
- Массив переменных вместо хеш-таблицы

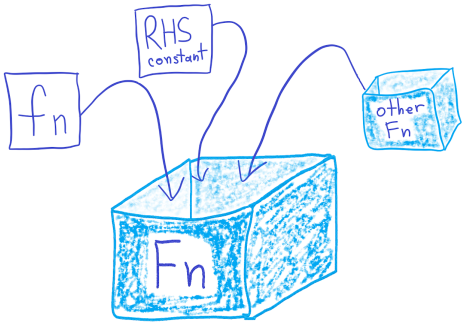

Рис. 2: Выражение в виде замыканий [1]

[1] https://blog.cloudflare.com/building-fast-interpreters-in-rust/

Сравнение подходов

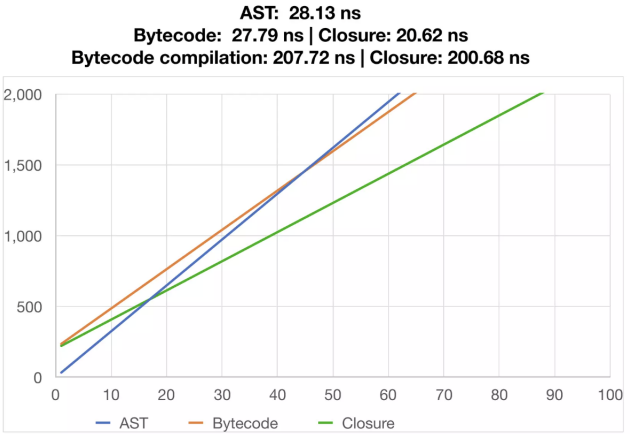

Рис. 3: Время исполнения выражений от количества прогонов[2]

[2] https://www.slideshare.net/slideshow/building-fast-interpreters-in-rust/125271384

Компоненты библиотеки исполнения правил

- Pratt Parser
  - Гибкость добавления новых конструкций
  - Детализированный вывод ошибок
- Builder
  - Преобразует узел AST в замыкание вида: func(P) Type
    - ⋆ P — интерфейс, предоставляющий доступ к окружению переменных во время выполнения
    - ⋆ Type — один из поддержанных типов: bool, string, float64, массивы базовых типов, any — для передачи пользовательских структур в функции
  - Проверяет типы и вычисляет узлы без переменных на этапе компиляции
- Value Provider
  - На основе хеш-таблицы предоставляет удобный доступ к переменным
  - На основе массива поддерживает более быстрый доступ к переменным, но пользователю нужно сохранить индексацию передаваемую в Builder
  - Переменные представляют собой функции, что позволяет вычислять только используемые переменные

Язык выражений

```
expr = ternary;

ternary = log_or
| log_or "?" expr ":" expr;

log_or = log_and
| log_or "||" log_and;

log_and = in
| log_and "&&" in;

in = bit_or
| in ("in" | "not in") bit_or;

bit_or = xor
| bit_or "|" xor;

xor = bit_and
| xor "^" bit_and;

bit_and = eq
| bit_and "&" eq;

eq = comp
| eq ("==" | "!=") comp;

comp = shift
| comp ("<" | "<=" | ">" | ">=") shift;

shift = add
| shift ("<<" | ">>") add;

add = mult
| add ("+" | "-") mult;

mult = pow
| mult ("*" | "/" | "%") pow;

pow = unary
| pow "**" unary;

unary = primary
| ("-" | "+" | "!" | "~") unary

primary = literal
| id
| func_call
| "(" expr")";

literal = boolean
| numeric
| string
| array;

boolean = "true" | "false";

numeric = decimal
| hexadecimal
| octal
| binary
| float
| e_notation;

decimal = digit+;
hexadecimal = "0x" hex_digit+;
octal= "0o" oct_digit+;
binary = "0b" bin_digit+;
float = digit+ "." digit+ | "." digit+;
e_notation = (digit+ | float) [eE] [+-]? digit+;
digit = "0"..."9";
hex_digit = digit | "a"..."f" | "A"..."F";
oct_digit = "0"..."7";
bin_digit = "0" | "1";

string = "\"" char* "\"";

func_call = id "(" [expr ("," expr)*] ")";
id = letter (letter | digit | "_")*;
letter = "a"..."z" | "A"..."Z";

array = "[" [expr ("," expr)*] "]";
```

Рис. 4: Грамматика языка

Тестирование и внедрение разрабатываемой библиотеки (feev)

- Каждый пакет эвалюатора покрыт юнит-тестами более чем на 85%
- Проведено сравнительное тестирование на синтетических данных ValueProvider на основе хеш-таблицы и массива, а также данной библиотеки с Expr и Govaluate
- Библиотека интегрирована на всех сервисах-агентах, которые используют правила для фильтрации трафика
  - На сервисах сэкономлено потребление 359 ядер CPU, 152 ГБ RAM
  - В среднем время проверки запроса сенсором ручного-рейтлимитера уменьшилось на 8%
- Проведено сравнение разработанной библиотекой и Expr на основе реальных данных и правил одного из агентов
  - Библиотека оказалась быстрее аналога в среднем в 2 раза

Бенчмарки проводились со следующими характеристиками устройства: CPU — Apple M1 Pro, 8 ядер, 3.2 GHz, RAM — 16 GB LPDDR5, macOS 14.4.1

Тестирование и внедрение разрабатываемой библиотеки (feev)

Test Case	Time (ns/op)
Simple expr	
map	165.0
func_slice	85.30
Multi_call contains func	
map	214.7
func_slice	155.2
Complex expr	
map	3093.0
func_slice	1918.0

Таблица 1: Сравнение производительности ValueProvider на основе хеш-таблицы и массива функций

Тестирование и внедрение разрабатываемой библиотеки (feev)

Test Case	Time (ns/op)	Memory (B/op)	Allocations/op
Simple expr without vars			
feev	0.9477	0	0
expr	46	0	0
goval	377	56	6
Simple expr			
feev	78	0	0
expr	1478	504	59
goval	4233	3208	356
fibonacci(five)			
feev	22	0	0
expr	79	24	2
goval	157	40	3
Multi_call contains func			
feev	152	0	0
expr	316	96	3
goval	547	184	7
Complex expr			
feev	1661	0	0
expr	46853	19440	1826

Таблица 2: Сравнение производительности библиотек feev, expr и govaluate

Результаты

- Определено положение библиотеки исполнения правил в антибот-системе
- Проведен анализ Golang-библиотек Expr и Govaluate, реализующих разные подходы к исполнению правил, с выявлением их недостатков и оценкой подхода на замыканиях
- Спроектированы ключевые компоненты библиотеки и их функциональные назначения
- Реализована библиотека исполнения правил на Golang, включающая:
  - Парсер на основе алгоритма Пратта для обработки DSL
  - Типизированный компилятор, преобразующий выражения в замыкания
  - Провайдер переменных на основе хеш-таблицы и массива
  - Синтаксис языка совместимый с текущими правилами антибот-системы
- Библиотека была покрыта модульными тестами на 85%, интегрирована в антибот-систему Ozon, а также проведен сравнительный анализ производительности библиотеки на основе синтетических и реальных данных, а также метрик сервисов

Исходный код закрыт, но в будущем планируется выложить в репозиторий «Ozon Tech»[3]

[3] `https://github.com/ozontech`

Компоненты библиотеки исполнения правил

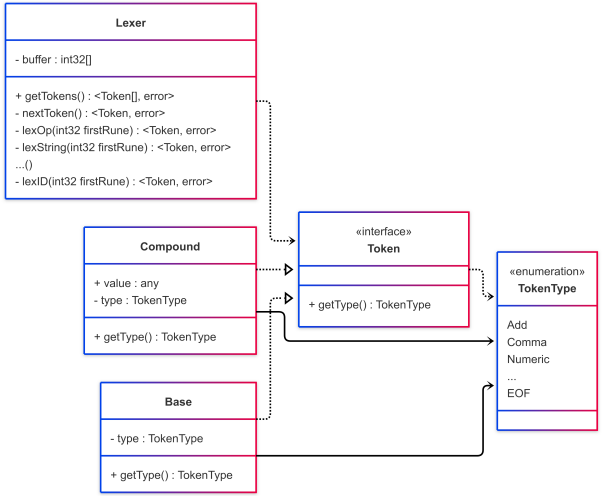

Рис. 5: Диаграмма классов лексера

Компоненты библиотеки исполнения правил

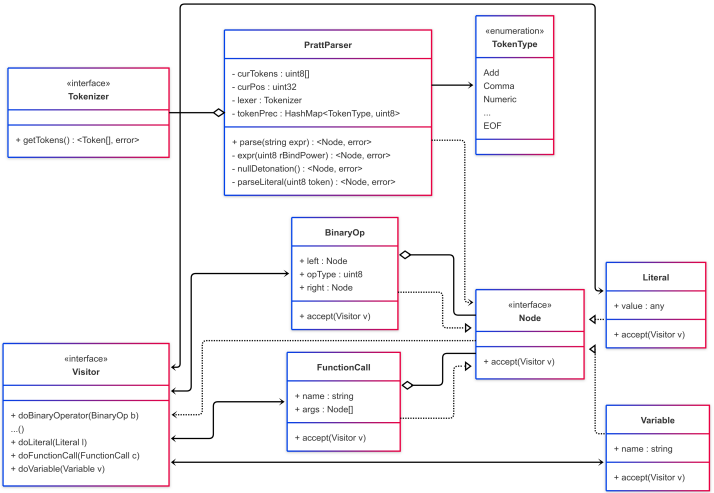

Рис. 6: Диаграмма классов парсера

Компоненты библиотеки исполнения правил

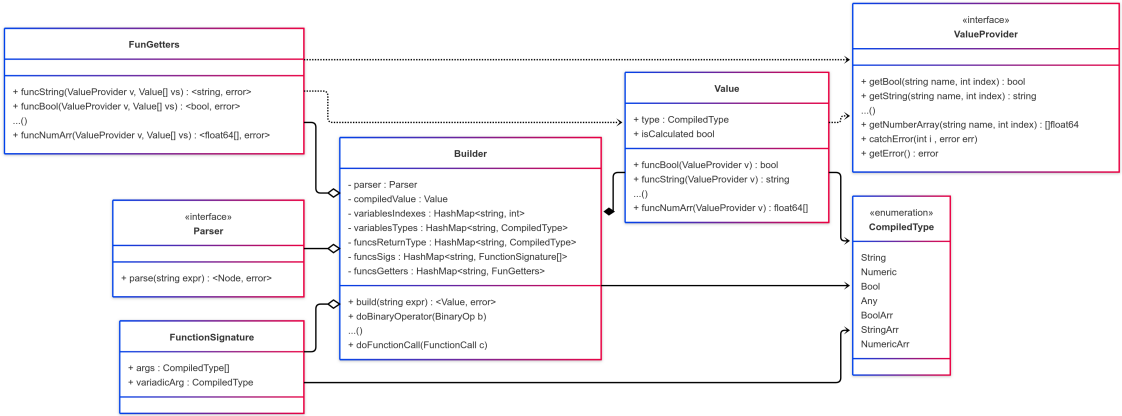

Рис. 7: Диаграмма классов компилятора

Компоненты библиотеки исполнения правил

Рис. 8: Диаграмма классов провадера переменных