rust-code-analysis分析报告

工具简介

rust-code-analysis 是一个由Rust编写的源码分析工具，能够分析并抽取源代码的相关信息，支持包括C++、Python、Javascript、Rust和Go在内的多种编程语言 。该工具主要功能为分析代码的度量指标，以及其它的一些源码操作小功能，例如删除指定文件中的注释。

源码|文档

工具功能

1. 该工具目前支持解析和分析下述11种编程语言：

- C++
- C#
- CSS
- Go
- HTML
- Java
- JavaScript
- The JavaScript used in Firefox internal
- Python
- Rust
- Typescript

2. 该工具目前支持下述代码度量计算（C#, CSS, Go, HTML, 和 Java尚未实现）

- **复杂度计算**: 计算源码的复杂度，包括认知复杂度和圈复杂度。
- **源码行数**: 统计源代码文件中的行数。
- **物理代码行数**: 统计源代码文件中物理行数，按换行符计算。
- **逻辑代码行数**: 统计源代码文件中逻辑代码行数。
- **注释行数**: 统计源代码文件中的注释行数。
- **空白行数**: 统计源代码文件中的空白行数。
- **Halstead复杂度**: Halstead 复杂度根据程序中语句行的操作符和操作数的数量计算程序复杂性。操作符通常包括语言保留字、函数调用、运算符，也可以包括有关的分隔符等。操作数可以是常数和变量等标识符。操作符和操作数的量越大，程序结构就越复杂。
- **可维护性指标**: 用来描述软件的可维护性。
- **NOM**: 统计 文件/trait/类 中函数或者闭包的个数。
- **NEXITS**: 统计函数或方法可能的出口个数。
- **NARGS**: 统计函数或方法的参数个数。

3. 除此之外，该工具还支持下述源码操作功能：

- 删除指定源代码文件中的注释
- 找出源代码文件中的语法错误并计数
- 打印源文件的AST
- 打印指定起始和结尾行数的AST

实现原理

rust-code-analysis之所以能够同时支持多种语言的源代码分析以及代码度量的计算，是因为使用了Tree Sitter这个源码解析库。Tree Sitter是一个C语言实现的，通用的源码解析器，它能够快速鲁棒的为各种编程语言生成抽象语法树AST。

- 认知复杂度计算：CognitiveComplexity
- 圈复杂度Rust实现：对于下述节点分支`If | For | While | Loop | MatchArm | MatchArm2 | QMARK(?) | AMPAMP(&&) | PIPEPIPE(||)` 圈复杂度加1。
- LOC实现: Tree Sitter解析生成的AST节点可以获取起始行数，再遍历节点然后判断节点类型来计算物理代码行数，逻辑代码行数和注释代码行数，空白行数通过总代码行数减去物理代码行数和注释行数来获取。
- Halstead复杂度实现原理：简单粗暴，需要给定操作数和操作符的数量，然后计算。
- 可维护性指标实现原理：简单粗暴，给定loc，Halstead复杂度和圈复杂度，然后计算。
- NEXITS, NOM和NARGS实现原理：略

工具使用

rust-code-analysis有三种使用方法，分别是库调用，命令行工具和Web接口调用。

- **rust-code-analysis-cli**: 打印代码度量示例`rust-code-analysis-cli -m -p /path/to/your/file/or/directory`
- **rust-code-analysis-web**：`rust-code-analysis-cli --serve --port 9090` 然后访问`http://127.0.0.1:9090/metrics?file_name={filename}&unit={unit}`

输出结果

输出结果可以直接打印到命令行，也可以保存为文件，支持下述导出文件格式：

- Cbor
- Json
- Toml
- Yaml