Дата последней сборки документа:
22 января 2023 г. в 13:26

Практика использования и наиболее полезные конструкции языка Rust

Содержание

1. Ресуры по языку Rust

https://www.rust-lang.org/tools
https://doc.rust-lang.org/book/
https://doc.rust-lang.org/stable/rust-by-example/

2. Установка Rust

Установить Rust проще всего с помощью утилиты `rustup` – это установщик языка и менеджер версий. Для операционной системы Windows можно скачать `rustup-init.exe` со страницы проекта https://www.rust-lang.org/learn/get-started

Установить Rust на Linux можно так

```
$ curl https://sh.rustup.rs -sSf | bash
...
Current installation options:


default host triple: x86_64-unknown-linux-gnu
default toolchain: stable (default)
profile: default
modify PATH variable: yes

1) Proceed with installation (default)
2) Customize installation
3) Cancel installation
>1

info: profile set to 'default'
```

```
info: default host triple is x86_64-unknown-linux-gnu
info: syncing channel updates for 'stable-x86_64-unknown-linux-gnu'
...
```

Rust часто обновляется и чтобы получить последнюю версию, можно воспользоваться командной `rustup update`.

Собрать проект и обновить его зависимости можно с помощью утилиты `cargo`

```
cargo build  # build your project
cargo run  # cargo run
cargo test  # test project
cargo doc  # build documentation for your project
cargo publish  # publish a libarary to crates.io
```

То есть `cargo` знает, как превратить Rust-код в исполняемый бинарный файл, а также может управлять процессом загрузки и компиляции проектных зависимостей.

3. Вводные замечания

Система владения устанавливает *время жизни* каждого значения, что делает ненужным сборку мусора в ядре языка и обеспечивает надежные, но вместе с тем гибкие интерфейсы для управления такими ресурсами, как сокеты и описатели файлов. Передеча (move) позволяет передавать значение от одного владельца другому, а заимствование (borrowing) – использовать значение временно, не изменяя владельца.

Rust – типобезопасный язык. Но что понимается под типобезопасностью? Ниже приведено определение «неопределенного поведения» из стандарта языка C 1999 года, известного под названием «C99»: *неопределнное поведение – это поведение, являющееся следствием использования непереносимой или некорректной программной конструкции либо некорректных данных, для которого в настоящем Международном стандарте нет никаких требований.*

Рассмотрим следующую программу на C

```
int main(int argc, char **argv) {
    // объявление одноэлементного массива беззнаковых длинных целых чисел
    unsigned long a[1];
    // обращение к 4-ому элементу массива; индекс, нарушает границу диапазона
    a[3] = 0x7ffff7b36cebUL;
    return 0;
}
```

Эта программа обращается к элементу за концом массива `a`, поэтому согласно C99 ее *поведение не определно*, т.е. она может делать все что угодно. «Неопределенная» операция не просто возвращает неопределнный результат, она дает программе карт-бланш на *произвольное выполнение*(!).

C99 предоставляет компилятору такое право, чтобы он мог генерировать более быстрый код. Чем возлагать на компилятор ответственность за обнаружение и обработку странного поведения вроде выхода за конец массива, стандарт предполагает, что программист должен позаботиться о том, чтобы такие ситуации никогда не возникали.

Если программа написана так, что ни на каком пути выполнения *неопределенное выполнение невозможно*, то будем говорить, что программа *корректна* (well defined).

Если встроенные в язык проверки *гарантируют корректность программы*, то будем называть язык *типобизопасным* (type safe).

Тщательно напсанная программа на C или C++ может оказаться типобезопасной, но ни C, ни C++ не является типобезопасным языком: в приведенном выше примере нет ошибок типизации, и тем не менее она демонстрирует неопределенное поведение. С другой стороны, Python – *типобезопасный* язык, его интерпретатор тратит время на обнаружение выхода за границы массива и обрабатывает его лучше, чем компилятор C.

4. Начало работы

Создать проект на Rust можно командной `cargo new <project_name>`

```
$ cargo new hello  # создать проект hello
$ tree
.
hello/
  Cargo.toml
  src/
    main.rs
$ cd hello
$ cargo run  # запустить проект
   Compiling hello v0.1.0 (/home/kosyachenko/Projects/GARBAGE/rust_projects/hello)
Finished dev [unoptimized + debuginfo] target(s) in 0.42s
Running `target/debug/hello`
Hello, world!
# Дерево проекта изменилось
$ tree
.
Cargo.lock  # артефакт
Cargo.toml
src/
  main.rs
target/  # артефакт
  CACHEDIR.TAG
  debug/
    build
    deps/
      hello-27...
      hello-27...d
    examples/
    hello
    hello.d
    incremental/
      hello-imy.../
        s-ghim...
```

В основном каталоге имеется файл `Cargo.toml`, содержащий описание метаданных проекта, таких как имя проекта, его версия и его зависимости. Исходный код попадает в директорию `src`.

Выполнение команды `cargo run` привело также к добавлению к проекту новых файлов. Теперь у нас в основном каталоге проекта есть файл `Cargo.lock` и каталог `target`. В `Cargo.lock` указываются конкретные номера версий всех зависимостей, чтобы будущие сборки составлялись точно также, как и эта, пока содержимое `Cargo.toml` не изменится.

4.1. Первая программа на Rust

Нужно как обычно с помощью `cargo new hello` создать новый проект. Перейти в созданную директорию проекта и в файле `main.rs` директории `src` написать следущее

./src/main.rs

```
fn greet_world() {
    println!("Hello, world!");
    let southern_germany = "Germany";
    let japan = "Japan";
    let regions = [southern_germany, japan];

    for region in regions.iter() {
        println!("{}", &region);
    }
}

fn main() {
    greet_world();
}
```

Восклицательный знак свидетельствует об использовании *макроса*. Для операции присваивания в Rust, которую правильнее было бы называть *привязкой переменной*, используется ключевое слово `let`. Поддержка Unicode предоставляется самим языком.

Для *литералов массива* используются *квадратные скобки*. Для возврата итератора метод `iter()` может присутствовать во многих типах. Амперсанд «заимствует» `region` так, чтобы доступ предоставлялся *только для чтения*.

Строки ганатировано получают кодировку UTF-8.

Список литературы

1. *Кольцов Д.М.* Си на примерах. Практика, практика и только практика. – СПб.: Наука и Техника, 2019. – 288 с.

Листинги