

使用Rust开发跨平台SDK



UGC前端组 冯伟

Trip.com Group™

2024/04/24

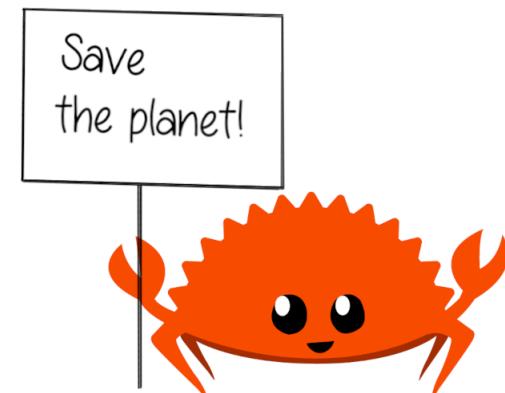
背景

在当今的软件开发领域，图像处理已成为一个不可或缺的部分。从简单的图片编辑到复杂的机器视觉应用，图像处理技术的应用场景日益广泛。然而，开发者在面对图像处理需求时，往往面临着以下困境：

1. 平台差异性: 不同的操作系统和设备有着不同的API和图像处理库，导致开发者需要为每个平台编写特定的代码。
2. 语言限制: 现有的图像处理库往往只支持特定编程语言，限制了开发者在项目中使用多种语言的能力。
3. 安全性和稳定性: 在图像处理中，内存管理和并发控制不当可能导致安全漏洞或程序崩溃

为什么选择Rust?

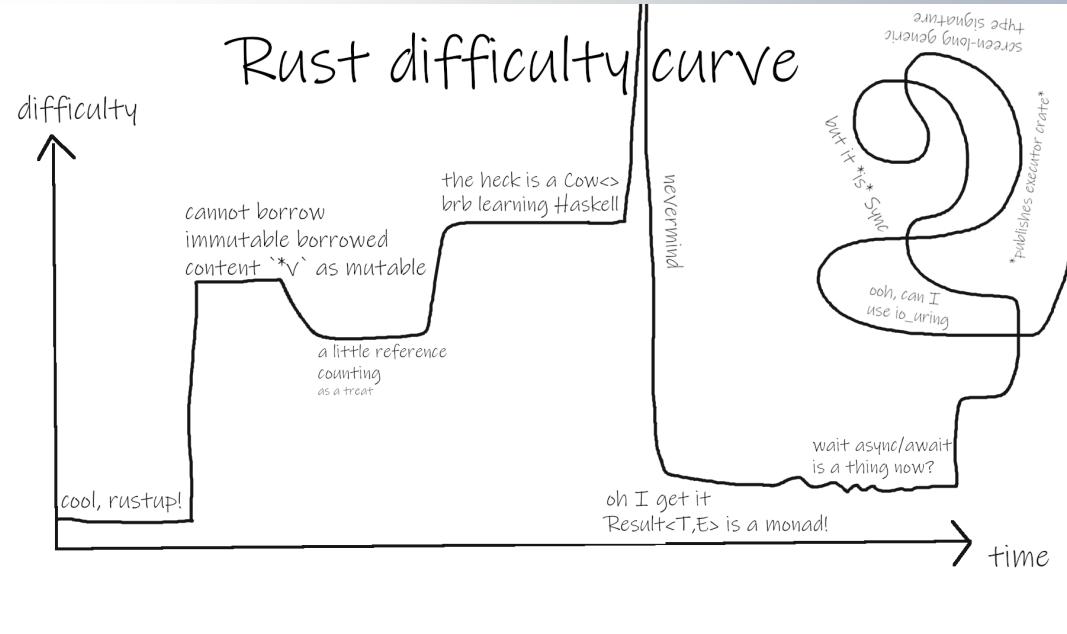
- 内存安全：Rust的所有权系统保证了内存安全，减少了内存泄漏和悬挂指针的风险。
- 并发性：Rust的并发模型简单而强大，可以在不牺牲安全性的前提下提高程序的执行效率。
- 跨平台：Rust支持跨平台编译，可以轻松构建适用于多种操作系统和CPU架构的应用程序。
- 性能：Rust的性能接近C和C++，适合对性能有较高要求的图像处理任务。
- FFI（外部函数接口）：Rust的FFI特性使得它可以轻松地与C语言互操作，进而支持在其他编程语言中调用。
- Save the planet, code in Rust 🐀



Trade off

选择Rust意味着:

- 范式转移 (Paradigm Shift)
- 大量计算机体系结构、操作系统知识扑面而来



Alternatives

1. **C/C++**: 虽然`C/C++`在性能上与`Rust`相当，但它们缺乏内存安全保障，增加了开发和维护的难度。
2. **Java**: `Java`具有良好的跨平台特性和成熟的生态系统，但其性能不如Rust，特别是在处理图像数据时。
3. **Go**: `Go`语言在并发编程和网络服务方面表现出色，但在系统级编程和与C语言互操作方面不如Rust灵活。
4. **WASI**: WebAssembly System Interface, 为WebAssembly定义了一套系统调用接口, 但其生态尚不完善.

Towards a modern Web Stack.

选择Rust是因为它在性能、安全性、并发性以及跨平台能力方面提供了最佳平衡，同时其FFI特性也满足了与多种编程语言互操作的需求。

从C调用Rust

```
#[repr(C)]
pub struct User {
    pub name: String,
    age: u8,
}

#[no_mangle]
pub extern "C" fn hello_from_rust() {
    println!("Hello from Rust!");
}
```

```
extern void hello_from_rust();

int main() {
    hello_from_rust();
    return 0;
}
```

```
$ rustc ./extlib.rs --crate-type=cdylib -C link-args="-s" -o libextlib.so
$ gcc ./main.c -o main -L . -lextlib
$ ./main
Hello from Rust!
```

从Rust中调用C函数

```
#[link(name = "extlib")]
extern {
    fn hello_from_c();
}
```

```
fn main() {
    unsafe {
        hello_from_c();
    }
}
```

```
#include <stdio.h>
#include <stdint.h>

void hello_from_c() {
    printf("Hello from C!\n");
}
```

```
$ gcc -shared -o libextlib.so extlib.c
$ rustc ./main.rs -l extlib -L .
$ ./main
Hello from C!
```

如何实现？

FFI: Rust的FFI特性使得它可以轻松地与C语言互操作，进而支持在其他编程语言中调用。

[gitlab/imagekit](#)

如何实现？

在构建`imagekit` SDK的过程中，精心挑选了一系列Rust生态中的库（crates），它们为项目提供了强大的功能支持和灵活性。以下是此项目中使用的主要技术：

- `image`: 提供了丰富的图像处理功能和格式转换能力，Rust在图像处理领域的事实标准库。
- `uniffi`: 为Kotlin、Swift、Python、Ruby等多种编程语言生成接口绑定。
- `prost`: ProtoBuf编解码。
- `tonic`: 高性能的gRPC框架，支持Rust的异步特性，为构建分布式系统提供了强大的支持。
- `salvo`: 快速且灵活的HTTP服务器框架，能够轻松处理各种HTTP请求。
- `schemars`: 生成`JSON Schema`，帮助定义和验证数据结构，确保数据的一致性和有效性。
- `clap`: 在命令行参数解析方面，clap提供了强大的功能，允许轻松构建复杂的命令行界面。
- `cxx`: 用于创建和使用C++绑定的库，使得Rust能够无缝地与现有的C++生态系统集成。
- `napi-ohos`: 使用Rust开发HarmonyOS的`Node-API`扩展

除以上核心库，还利用了`wasm-bindgen`和`js-sys`来支持在WebAssembly环境中调用imagekit。这使得SDK能够在现代Web浏览器、Wasmer等环境中运行，极大地拓展了其应用范围。此外，还使用了`quicktype`来生成多种编程语言的数据模型。

未来展望

- 通过Dioxus/TAURI/egui开发跨多端应用/Desktop、Mobile、Web) [Are we GUI Yet?](#)
- 跨平台业务逻辑复用
 - crux
 - xstate-rs
- 构建跨平台组件
 - ndk: Rust bindings for the Android NDK
 - cocoa: Provides safe bindings for `AppKit` and `UIKit`
 - HarmonyOS Native XComponent

参考

- [Firefox Application Services](#)
- [Bitwarden SDK](#)
- [使用Node-API实现跨语言交互](#)
- [FeatureProbe](#)

END