黑客派

使用 Rust + Electron 开发跨平台桌面应用(二)

作者：leoython

原文链接：https://hacpai.com/article/1554611902061

来源网站：黑客派

<h3 id="前言">前言</h3>
在上一篇文章使用 Rust + Electron 开发跨平台桌面应用 (一)中，我们将 Rust + Electron 结合起来，使用 Rust 编写核心业务辑，并编译成 node 库提供给 Electron 的 UI 界面调用，但是在上一篇文章中发现遇到了很多问题，其是 Electron 的版本和 Rust 编译出来的版本必须要一致，否则会无法调用成功，这就很坑了，所以了改变这一情况，今天我们将使用另一种方式将 Rust 的代码提供给 Js 进行调用，这就是 FFI。
<h3 id="FFI是什么">FFI 是什么</h3>
FFI(Foreign Function Interface)是用来与其它语言交互的接口，由于现实中很多程序是由不同编程语言写的，必然会涉到跨语言调用，这时一般有两种解决方案：
1、将函数做成一个服务，通过进程间通信(IPC)或网络协议通信(RPC, RESTFul 等)；
2、直接通过 FFI 调用。
前者需要至少两个独立的进程才能实现，而后者直接将其它语言的接口内嵌到本语言中，所以调效率比前者高。
Rust 作为系统级编程语言，也是对 FFI 提供了完善的支持。
<h3 id="mangle">mangle</h3>
由于 rust 支持重载，所以函数名会被编译器进行混淆，就像 c++ 一样。因此当你的函数被编译毕后，函数名会带上一串表明函数签名的字符串。
 这样的函数名为 ffi 调用带来了困难，因此，rust 提供了 #[no_mangle]属性为函数修饰。 对于带有 #[no_mangle]属性的函数，rust 编译器不会为进行函数名混淆， 如：
<pre><code class="highlight-chroma">#[no_mangle]
pub extern fn test() {}
</code></pre>
下面我们来编写一个 thread_count.rs，其实跟寻常的 rust 代码没有什么区别：
<pre><code class="highlight-chroma">#[no_mangle]
pub extern fn threadcount(x: i32) -> i32 {
    let result: i32 = num_cpus::get() as i32;
    return result * x;
}
</code></pre>
<h3 id="指定库类型">指定库类型</h3>
rust 默认编译成 rust 自用的 rlib 格式库，要让 rust 编译成动态链接库或者静态链接库，需要显指定，一共有三种方式，我这里采用的是直接在 Cargo.Toml 文件中指定，如下：
<pre><code class="highlight-chroma">[lib]
name = "thread_count"
crate-type = ["dylib"]
</code></pre>
需要注意的是 <code>name</code>，必须符合 rust 的包结构，能够在 src 目录下找到。

我们执行 cargo build 命令，可以看到，在/target/debug 目录下生成了我们需要的文件 libthread_count.dylib
<img src="https://static.hacpai.com/images/img-loading.svg" alt="clipboard.png" data-sc="https://image-static.segmentfault.com/318/311/3183115311-5c529b55c3828_articlex">
<h3 id="JS使用rust的动态链接库">JS 使用 rust 的动态链接库</h3>
那么我们要如何在 JS 中调用 rust 生成 dylib 呢？答案就是 ffi-napi，我们使用 ffi-napi 这个包在 js 中调用 ffi，话不多说，直接看代码
<pre><code class="highlight-chroma">let ffi = require('ffi-napi');
let path = require('path');

let threadCount = ffi.Library(path.join(__dirname, './target/debug/libthread_count'), {

```
threadcount: ['int', ['int']]
});

let result = threadCount.threadcount(12);
console.log("thead_count: " + result);
</code></pre>

<p>结果如下：</p>
<p></p>
<p>好了，到此为止，我们就成功的将 rust 编译成动态链接库给 JS 调用了，这种方式是我觉得比较的一种方式，虽然引入函数的方式比较丑，但是我们不用担心 node 版本的问题。</p>
<h3 id="结语">结语</h3>
<p>虽然 FFI 是一种我认为比较好的方式，但是它也不是完美无缺的，例如，在跨越 FFI 的过程中，们会丢失 rust 的类型信息，从而引发安全性问题，当然这也不是没有解决办法，我们可以使用 rust 的 Box 来包装我们的类型，这个可以单独开一篇文章来讲述，就不展开了（先挖个坑，哪天想起来再填</p>
```