# Angular调用C/C++函数

在 NodeJS 中，和C/C++语言编写的代码通信主要有两种方案：

* 使用 AddOn 技术，使用 C++为 NodeJS 编写一个拓展，然后在代码中调用其他语言所编写的源码 or 动态库。addon是nodejs官方的C++扩展实现方案，但是由于需要使用模版，并且要对v8引擎有一定的了解，入门门槛较高。
* 使用 FFI（Foreign Function Interface）技术，直接在 Nodejs 中引入其他语言所编写的动态链接库，ffi是nodejs直接调用so库的一种实现，可以调用纯c的接口。

【注意】FFI只接受纯C函数, 确切的说, 是按照C标准编译的函数。

这是因为C和C++编译后的函数名字是不同的,。

对于C, 函数int func(int n)会被编译为类似\_func这样的名字.

对于C++, 函数int func(int n)会被编译为类似?func@@YAHH@Z这样的名字.

同样是C++, 函数int func(int double)会被编译为类似?func@@YAHN@Z这样的名字(和上一个不同)。名字中包含了较多信息, 比如:参数的入栈方式、返回值的类型、参数的类型和数量。

这是因为C++有函数重载特性, 虽然函数命名是func, 但int func(int n)和int func(int double)完全是两个不同的函数, 编译器通过给它们赋予不同的名字来区分它们。回到FFI, 它在dll中查找函数名字的时候, 是用C风格来查找的。.

所以如果你的函数使用C++编译的, ffl在这个dll中就找不到这个函数, 会提示错误LINK 126!

AddOn 技术比较通用，它可以使用 C++代码来拓展 NodeJS 的行为，很多库都是使用这种方式来完成一些比较底层的操作（比如和操作系统的一些通信）。但是它写起来比较麻烦，要编写一个 C++项目，还要按照 NodeJS 的规范 export 相应的函数，而且每次安装的时候都需要进行编译（以适应本地 Node 的版本）。如果只是调用一个 DLL，那就还需要在项目里重新包装一遍 DLL 的接口。

如果使用 FFI 技术，限制就会比较多。首先，它只能调用其他动态库，如果你想使用 C/C++完成更多功能的话，还需要再封装一层 DLL，另外，它只支持\_cdecl调用约定（也就是 DLL 在导出的时候一定要标记用\_cdecl编译命令），不支持\_stdcall或者\_fastcall调用。但是调用起来就会很方便，可以直接在 JS 代码中声明 DLL 的接口就可以了。

综上比较，如果只调用第三方 DLL（而且恰好是\_cdecl导出），使用 FFI 就再合适不过了（虽然性能可能会有一定的损失，而且调试起来会有困难）。

其实，从理论上来讲，FFI 也是基于 AddOn 技术的，只是它可以帮你把在 JS 中定义的接口直接转换成 C 语言的接口，并利用 NodeJS 的 Buffer 内存，将其同载入的 DLL 共享。当然由于 FFI 的这种通用性，也导致了一定的性能损失。

## windows使用FFI调用动态库

### 1.安装nodejs

如果已经安装了最新版本的nodejs，在安装 FFI 的时候会出现各种兼容性的问题（比如编译无法通过，虽然已经有人提供了 patch，但是还没有被 merge 进主分支，为了避免出现 bug，还是暂时不用为妙）。可以安装 LTS 版本代替。

另外，还需要注意要调用的 DLL 是 32 位还是 64 位的，Node 的版本需要和 DLL 的版本匹配。因为如果 64 位 Node 调用了 32 位的 DLL，是无法成功装载的，反之亦然。

### 2.安装 Windows 的 C++工具链

这里有两种方案：

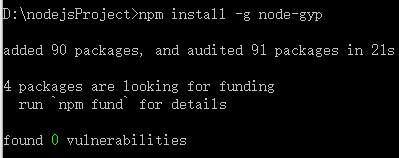
* 安装 Visual Studio，并安装相应的工具链。如果使用 VS 2019 版本的话，需要安装 C++桌面开发和 Windows SDK 相关的工具（Node v10 现在只支持 v141 版本的 MSVC），这种方式便于后续的调试工作（虽然也很艰难）
* 在安装 Node 之后，使用管理员权限运行 Powershell，并全局安装 windows-build-tools，参考命令 npm install --global --production windows-build-tools

### 3.安装node-gyp

node-gyp 是一个 Node 中基于 gyp 的跨平台的编译工具，用于编译其他库。

在安装的时候，需要使用 VC 的工具链，所以如果没有把工具链放在全局变量中，需要打开 VS 的Developer Powershell安装，该命令行一般在开始菜单的 Visual Studio 文件夹中。

参考命令：npm install -g node-gyp



### 4. 安装 FFI 及 REF

下面的步骤依旧需要 VC 工具链，所以可能依旧需要在Developer Powershell或者cmd中执行（建议常备该窗口，后面只要涉及到编译安装的命令都需要用到）。

安装 FFI 及相关工具的时候如果没有 VC 工具链，则会直接安装二进制代码，这样可能会出现包的 ABI 版本和 NodeJS 的 ABI 版本不符合的情况（在下面的 Tips 中会提到）。

现在，切换到项目的文件夹中，安装下面的包。其中，ffi 包是用以支持 FFI 功能的，ref 包是用以支持指针功能（原理是通过 Node 的 Buffer 内存，将 JS 的结构和 C 结构相互转换的）的，ref-\*是用以支持高级结构的（比如数组和结构体）

npm install ffi -s

npm install ref -s

npm install ref-array -s

npm install ref-struct -s

除此之外，如果想支持 VC 中常见的 wchar 类型，还可以安装 ref-wchar 包。

### 5.安装 electron-rebuild 包

如果是 electron 项目，还推荐安装 electron-rebuild 包，该包可以遍历 node\_modules 目录下的所有包，并将其重新编译。

然后，推荐在 package.json 中配置 electron-rebuild 的命令：

"scripts": {

"rebuild": "./node\_modules/.bin/electron-rebuild"

}

之后在需要重新编译的时候只需要执行npm run rebuild即可。

### 6.使用方式

如下官方示例：

var ffi = require('ffi')

var libm = ffi.Library('libm', {

ceil: ['double', ['double']]

})

libm.ceil(1.5) // 2

在引入 FFI 后，使用 FFI 调用了 libm 库（这个示例只能在类 Unix 系统中使用），一般拓展名为 libm.so，系统会在系统目录下搜索这个动态库，并将它使用动态链接器载入到 node 进程中。

接着，程序声明了 libm 库中的一个方法 ceil（向上取整），其中，该函数的返回值是 double 类型（第一个数组中的 doule），而该函数的入参也是一个 double 类型的值（第二个数组中的 double）。

最后，直接使用libm.ceil方法即可调用动态库中的函数，并返回正确的值。

## Linux下使用FFI调用动态库

### 1.安装nodejs

直接使用已编译好的包

Node 官网已经把 linux 下载版本更改为已编译好的版本了，我们可以直接下载解压后使用：

# wget https://nodejs.org/dist/v10.9.0/node-v10.9.0-linux-x64.tar.xz // 下载

# tar xf node-v10.9.0-linux-x64.tar.xz // 解压

# cd node-v10.9.0-linux-x64/ // 进入解压目录

# ./bin/node -v // 执行node命令 查看版本

v10.9.0

解压文件的 bin 目录底下包含了 node、npm 等命令，我们可以使用 ln 命令来设置软连接：

ln -s /usr/software/nodejs/bin/npm /usr/local/bin/

ln -s /usr/software/nodejs/bin/node /usr/local/bin/

### 2.安装node-gyp

npm install node-gyp -g

### 3.安装FFI

npm install ffi -S

### 4.安装ref

npm install ref -S

### 5.调用动态库

const ffi = ffi.Library('./demo.so', {

// 第一个 int：返回结果的数据类型， 第2,3个int 入参的数据类型

// 数据类型都是在C函数中定义好的

add:['int', ['int, int']]

})

const res = ffi.add(10,10)

console.log(res) // 20