Haskell FFI 小叙

Felis sap...
函数式编程、编程语言、编程 话题的优秀回答者

已关注

圆角骑士魔理沙、刘雨培、霜月琉璃、MaskRay、草莓大福等 52 人赞了该文章

最近做了不少与 Haskell 的 FFI 相关的实验，目的是为了做几个 C/C++ 库的绑定，以及做帮我做绑定不那么麻烦的工具。在这期间发现相关文档在网上比较分散，不方便 Haskell 初学者学习使用 FFI 调用外部库。希望这篇笔记能够有所帮助。

FFI 相关的 type 和 class

首先，Haskell 2010 Language Report 规定了一些 FFI 相关的 type 和 class 。

Storable class：这个 class 的主要方法包括 sizeOf、alignment、peek、poke。Storable class 的实例类型是“需要暴露给 C 函数的 Haskell 类型”，因此需要提供这些类型的 size/alignment，以及怎样在 Ptr 类型的指针指定的位置进行读写。一些在 C 里有直接对应版本的“基本” Haskell 类型（比如 Int、Double、Ptr 之类）已经是 Storable 的实例，而用户也可以针对自定义的 Haskell 类型去实现 Storable 实例，比如 record 可以用 struct 实现，多构造器的数据类型可以用 tagged union 实现，递归的数据类型可以用指针实现。这里的 peek/poke，可以直接当作 binary parser/serializer 来使用（只要有自信不怕 segfault），一个新近的序列化库 store 就是这么设计的，通过舍弃跨平台性来提高序列化性能。

Ptr type：这就是我们耳熟能详的 C 指针了。写 C 时那些与指针相关的 traps & pitfalls 自然一个不缺，比如 dangling pointer 和 memory leak。Ptr 类型带有一个类型参数，代表指向位置的类型（当然，我们也有 castPtr）

FunPtr type：对应 C 里函数指针的类型。普通的 Ptr 主要操作为 peek/poke，而 FunPtr 主要操作则是用于将 C 函数指针转为可以在 Haskell 里调用的函数，后面会介绍用法。

个指针 unreachable 以后自动 free，岂不美哉？所以有了 ForeignPtr 类型：可以通过一个 Ptr 以及一个 finalizer （销毁 Ptr 所占资源的 IO callback）构造出来，之后程序员便无需操心何时应该调用 finalizer 的问题。

foreign import / export 声明

理解了 FFI 相关的几个 type 与 class 的用法以后，我们可以使用 foreign import 导入 C 函数，foreign export 导出 Haskell 函数。

foreign import 声明的示例如下：

```
foreign import ccall safe "string.h strlen"
    c_strlen :: CString -> IO CSize
```

这个声明包含几个元素。ccall 是调用 C 函数所需的调用约定；如果是 32-bit 下调用 Windows API，则使用 stdcall 。后面的这个 safe 一项是可选的，指的是 safety level，safe 与 unsafe 相比，每次调用会有少许的 overhead，但是 unsafe 的调用中不允许外部代码调用 Haskell，而且有风险将整个 Haskell 运行时给 block 住（例）。这一项省去的话默认是 safe。

接下来的内容用于指定 import 对象的符号，以及在 Haskell 命名空间内的名字和类型签名。在格式串中，header name 可以省略，只要 GHC 在 linking 阶段能够从符号表中找到对应符号即可。如果整个格式串略去，则默认符号名与 Haskell 绑定同名。类型签名是强制的，如果是带副作用的函数，最后返回值应该是 IO t，纯函数则可以返回 t。GHC 不会检查 foreign import 的类型是否与 C 类型契合（符号表里也没这个信息）。如果格式串中，在符号名前加一个 &，比如 "&some_static_var"，则代表一个 static variable 而非 static function，此时可以将其 import 为 Ptr t 的形式，获得指针并进行读写。

在 foreign import 生成的函数中，所有参数类型和返回类型必须是 marshallable foreign type，简而言之，限定为 Bool、Char、Int、Double、Ptr 等一系列类型，及其 type synonym 和 newtype 的范围以内，不支持（哪怕实现了 Storable 实例的）用户自定义数据类型。这是一个比较大的限制，如果需要绑定的 C 库中，有直接将 struct 等构造传值的函数，我们需要首先在 C 这边写一个传指针的 wrapper 函数，再对这个 wrapper 函数做 import 。

以上的 static import 针对的是链接库中地址已知的 static function/variable，另外还有针对函数指针的 dynamic import 和 dynamic wrapper 两类 foreign import。dynamic import 的示例如下：

```
foreign import ccall safe "dynamic"
    mkFun :: FunPtr (CString -> IO CSize) -> CString -> IO CSize
```

这类函数的参数是一个 C 函数指针，将其转为一个可在 Haskell 中进行调用的函数。

而 dynamic wrapper 做的则是反方向的事情，将一个可在 Haskell 中进行调用的函数转为 C 函数指针，然后传给一个需要 C callback 的 C 库函数：

```
foreign import ccall safe "wrapper"
    mkCallback :: IO CInt -> IO (FunPtr (IO CInt))
```

除了 foreign import 以外，也有 foreign export 声明，用于将 Haskell 的函数导出到头文件中，允许 C 进行调用。foreign export 的语法与限制与 import 类似，export 以后，GHC 会在编译中生成一个 header，可以在 C 中 #include 并调用相关函数，具体参考 GHC 文档中相关部分。

capi 调用约定

调用 C 库中的函数使用的调用约定是 ccall，它要求在链接库的符号表中找到对应的符号。不过有时候我们希望 import 的一个常量或者“函数”是通过 C 预处理器语法给 #define 出来的，自然不会出现在符号表里。这种东西怎么 import 呢？GHC 提供了一个 CApiFFI 语言扩展，这样一来就可以使用 capi 调用约定，import 这些预处理器定义了。GHC 在编译时，普通的 ccall 调用，会通过平台的 ABI 处理，而对于 capi 调用，则会动态生成一个 C 源代码，#include 相应的头文件，并对每个 capi 调用生成一个 wrapper 函数，这样就可以将“假”的函数/值变成真的。使用 capi 调用约定时，必须提供相应的头文件名。

InterruptibleFFI 扩展

Haskell 2010 中，对 foreign import 可以指定 safe/unsafe。GHC 的 InterruptibleFFI 语言扩展提供了另一种 safety 选项：interruptible，interruptible 在 safe 的基础上增加了“通过平台相关的系统调用干掉一个 blocking I/O 动作”的特性。这个扩展与 Haskell 的异步异常特性有关，之后会进一步介绍。

关于线程模型与异常特性

在 Haskell 与 C 互操作时，我们需要对 Haskell 运行时的线程模型有基本的了解。Haskell 运行时有单线程与多线程（编译时使用 -threaded 选项）的版本，如果要在多线程 Haskell 程序中使用 FFI，不想每次 foreign call 挂起整个程序的话，必须使用多线程运行时。

多线程运行时会维护一个 OS thread pool，每个 OS thread （在 Haskell 这边称为 Capability）可能会负责许多不同 Haskell thread 的执行。Haskell thread 分为两种：unbound thread 和 bound thread，其中 bound thread 被绑定到一个特定的 Capability 上进行执行，调度的开销较高，unbound thread 则允许被调度器在不同 Capability 之间迁移，相对而言开销更低。

Control.Concurrent.forkOS 手动生成 bound thread，另外 Main.main 默认也是一个 bound thread。如果需要在 unbound thread 中进行此类 FFI 调用，频繁 forkOS 开销较大，则可以构造一个 channel （使用 MVar 或者 Chan），将执行 FFI call 的 IO action 通过 channel 送到一个 bound thread 执行，结果通过另一个 channel 返回。

关于异常特性，我们需要了解同步异常与异步异常。异步异常是通过 Control.Concurrent.throwTo 在其他 Haskell 线程中手动生成的异常，一般而言，我们进行 Haskell 的异常处理时，会 handle 同步异常，但是异步异常代表这个计算应该无条件立即终止，所以不会 handle 它，典型的例子是 async 库。当一个 Haskell thread 进行 blocking FFI call 时，哪怕异步异常也不会在其返回之前生效，因此我们可以通过 InterruptibleFFI 扩展，这样一来，Haskell 运行时会通过调用平台相关的系统调用，将进行中的 FFI call 强行终止，保证系统的可响应性。

关于内存分配

向 C 函数传递数据时，我们经常需要手动分配内存，并传递指针。分配内存的方法分为2类：

- 调用 malloc/calloc。这个分配行为由 C 运行时进行管理。
- 调用 alloca/mallocForeignPtr。这个分配行为由 Haskell 运行时管理。

绝大多数情况下，应该选择由 Haskell 运行时管理的分配方法。它们会在 Haskell heap 上分配一个 pinned ByteArray#，单次分配的开销远小于 C 运行时提供的实现。对于指针的生命周期明确，清楚何时析构的场合，使用 alloca；对于指针的生命周期不明确，需要 GC 帮忙自动析构的场合，使用 mallocForeignPtr。

如果不希望使用 GC 提供的自动析构，还有一个折衷的方法是使用 resourcet 或 io-region 辅助管理指针。在分配内存时，可以同时获取一个 Key，这个 Key 是析构器的句柄，可以在之后手动启动析构，并且可以多次调用，而在 IO action 退出当前的 region 以后，也会自动析构，这一点与 C++ 的 RAII 有一定类似。

辅助实现 C bindings 的工具

前面已经介绍与 FFI 相关的一些接口与概念，现在我们可以开始愉快地实现一个库的绑定了！

才怪。稍微写上规模大一点的绑定就会很快明白：

- GHC 不会检查 foreign import 的类型，写错了很容易导致莫名其妙的运行时错误
- GHC 不会帮你算 size 和 alignment
- 工作量太大了。一个用户友好的绑定库一般分为 3 层：底层是一些 C 代码用于在 C 这边做一些 Haskell 这边不方便的微小工作；中层是 C 定义与 Haskell 定义之间 1：1 对应的 foreign

用 reader monad 封装上下文信息，等等。

我们需要一些工具减少我们写 C bindings 的痛苦。这里简单列举一下我们用得上的东西：

- C 预处理器。别笑，我是认真的。开了 CPP 语言扩展以后我们可以直接在 Haskell 源代码里使用 C 预处理器语法了，这是很丑陋但却 work 的减少 boilerplate 的一大利器
- hsc2hs。这是一个 GHC 附带的 Haskell 预处理器，除了 C 预处理器语法以外，支持一组扩展的语法，可以用于生成 struct 的 peek/poke、转换简单类型、定义 enum 等等
- c2hs。这是另一个 Haskell 预处理器，有 Cabal 支持，可以从 C 头文件定义中生成 foreign import 与相关的 Haskell 类型
- inline-c。库如其名，通过 TemplateHaskell 和 QuasiQuotes 扩展，可以直接在 Haskell 代码中嵌入 C 代码片段，自动生成 C 代码并编译、链接，进行调用，免去手动 import 的麻烦。

动态链接和调用 C 函数

如果有“同一个绑定，链接不同版本的库”的需求，或者更干脆地，对于要绑定什么库、函数的名字和签名都在运行时获取的情况，Haskell FFI 是没有直接支持的，我们需要自己费点心思。

首先需要加载动态链接库、查找符号并拿到函数指针。Windows 下有 LoadLibrary / GetProcAddress，Unix 下有 dlopen/dlsym，这两个调用在 Win32/unix 两个库里都有绑定可以用的。拿到 C 函数指针以后，需要当成 Haskell 函数调用，我们有几条路可以走：

- 函数签名已知。这样的话，可以用前面提到的 dynamic import，生成的函数传入我们拿到的函数指针，获取可调用的 Haskell 函数
- 函数签名未知。GHC 帮不了我们了。不过有一个叫 libffi 的好东西，给他一个函数指针，以及运行时获取的函数类型信息，就可以按照这个信息传一系列参数（反正都是从 void* 来读）调用函数，而且支持不同 ABI。libffi 也有 Haskell 绑定
- 用 libffi 不久，我发现这货有缺陷：对 C 类型支持不完整啊。union 不支持，带 bit-field 的 struct 不支持，带 array 成员的 struct 也不支持。虽然是可以 hack 一下将就用，不过我已经弃用 libffi，改实现一个“poor man's JIT”来代替了：对于运行时才能获取类型信息的一个函数，生成一个“传入函数指针和参数指针数组，再进行调用”的 wrapper C，然后用 C 编译器编译以后动态链接进来，再把函数指针传给这个 wrapper 函数。这样，在 Haskell 中只需要一个 dynamic import 就可以搞定所有情况了

连接 C 以外的语言

- GHCJS 实现了 javascript 调用约定，可以调用 JavaScript 内建函数和 DOM API
- tweag.io 实现了 HaskellR，和 inline-c 类似，可以通过一组 quasi-quoters 直接在 Haskell 里写 R 代码做数据分析
- tweag.io 也实现了 inline-java，可以通过 JNI 访问 Java 上的对象与方法。另外还有 Eta，直接在 JVM 跑起来的 Haskell dialect，可以牺牲兼容性以省去 JNI 的开销

Welcome to the GHC Users Guide

Parallel and Concurrent Programming in Haskell

haskell.org/onlinerepor...

Archives : Inside 214-1E

预祝大家五一劳动节快乐。

#EOF

编辑于 2017-04-06

「真诚赞赏，手留余香」

赞赏

1 人已赞赏

Haskell 函数式编程 GHC（编程套件）

▲ 赞同 52 ▼ 18 条评论 分享 收藏 ···

文章被以下专栏收录

不动点高校现充部

一切与编程语言理论、函数式编程相关的杂谈。

已关注

推荐阅读

学 H
几年
滴ler
G
P

圆角骑士魔... 发表于雾雨魔法店 mirone 寸光寸阴 发表于Haske... Cosr

18 条评论

切换为时间排序

写下你的评论...

dram 1 年前

> 而且有风险将整个 Haskell 运行时给 block 住（例）。
都不用去试风险了，unsafe ffi call 总是会把这个 Haskell 线程所在的系统线程（capability）上的所有 Haskell 线程都阻塞住。
safe 的话就会把这个 capability 上的线程都扔回给 scheduler，可以在其它 capability 上跑，再去调用
所以你要是 GetMessage 或者 select 啥的会 block 比较长时间的东西或者你希望在 ffi call 的时候别的 Haskell 线程能运行的话，就 safe 吧（但是你既然有轻量级线程了还 select 啥。。。）

1

以上为精选评论

李约瀚 1 年前

咋就五一劳动节了(现在日期4月4日{}

赞

圆角骑士魔理沙 1 年前

投稿不（

赞

罗宸 1 年前

卧槽，ffi坑这么深...

赞

parker liu 回复 罗宸 1 年前

FFI要用好需要好的C基础，另外对操作系统要有了解，有一定的系统编程的能力。

赞 查看对话

neo lin 1 年前

这个坑踩的好

傻笑 1 年前

死程和死宅是不是存在某种必然性的联系...

1

霜月琉璃 1 年前

露娜Sama！啊，我死了

5

非洲鹞 1 年前

冲着露娜进来发现进错地方了

1

矢泽妮可 1 年前

为露娜而来，却被无情的程序打得我头昏

赞

Gao Fan 1 年前

题图满分

赞

Liutos边撸管边 1 年前

题图社保

赞

Felis sapiens (作者) 回复 Liutos边撸管边 1 年前

四斋蒸鹅心（

赞 查看对话

soiamsoNG 1 年前

haskell-gi X vala

赞

Felis sapiens (作者) 回复 soiamsoNG 1 年前

haskell-gi 的问题是 haddock 不友好（

赞 查看对话

阅千人而惜知己 1 年前

熊起 1 年前

对于带虚拟机的脚本语言，ffi将脚本语言原生函数转换为C回调函数的一般做法是什么？是不是临时生成一个函数，把原生函数和虚拟机指针作为写进代码段，C就是进去调用这个原生函数？

赞

Felis sapiens (作者) 回复 熊起 1 年前

chiark.greenend.org.uk/...

赞 查看对话