Interoperate Swift with C

关于C中的字符串指针

(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/252)

服务端的Socket demo - II

(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/254)

服务端的Socket demo - I

Back to series (/series/interoperate-swift-with-c)

在了解了所有Swift和C的交互细节之后，这一节，我们将使用Swift编写一个UNIX Domain stream socket，以此加深对之前讲过内容的理解。实际上，当你了解了如何在Swift中使用指针，以及如何把Unicode对应到C的字符串处理之后，你就会发现，把Swift作为一种系统编程语言的感觉，也还不错。

当然，我们的重点仍旧在Swift，所以，如果你还不熟悉Socket，建议还是先去给自己补补课。为了让这个例子尽可能简单，我们选择使用Socket实现了本地进程间通信，避免了转换网络地址以及字节序的麻烦。

大家可以在这里 (https://github.com/puretears/socket-demo-in-swift/tree/master/SocketSrv)找到服务端的完整实现。

服务端的实现

初始化Socket对象

在开始具体的编码前，不妨先想象一下，如果Swift版本的Socket编好了，应该怎么用呢？首先，是创建 `socket` 对象的部分，一个大致的想法，是这样的。在*main.swift*中，添加下面的代码：

```
do {
    let socket = try Socket(
        socketFilePath: "/tmp/swift_sock_demo",
        type: .passive)
}
catch {
    print(error.localizedDescription)
    exit(EXIT_FAILURE)
}
```

由于我们实现的是本地的进程间通信，因此，通过一个文件来共享数据就可以了。在 `Socket` 的 `init` 方法里，通过 `socketFilePath` 指定了这个文件，参数 `type` 表示创建的 `socket` 的类型，由于我们在编写服务端程序，因此，是 `.passive` 。如果创建失败了，我们就让 `init` 抛出各种异常，在这里，简单期间，我们只是捕获了所有异常，打印了对应的错误描述后退出程序。

当然，现在这段代码还不能通过编译，这只是我们想象出来的样子，为了让它可以工作，我们就参照着这份想象，先来实现 `Soket` 的 `init` 方法。

首先，在项目中，新建一个 `Socket.swift` ，并添加下面这些代码：

```
enum SocketException: Error {
    case cannotCreateSocketFile
    case socketPathTooLong
    case socketFileAlreadyExists
    case cannotBindSocketAddress
    case cannotListenOnTheSocketAddress
    case cannotAcceptConnection
    case cannotConnectToSocket
}

enum SocketType {
    case active
    case passive
}
```

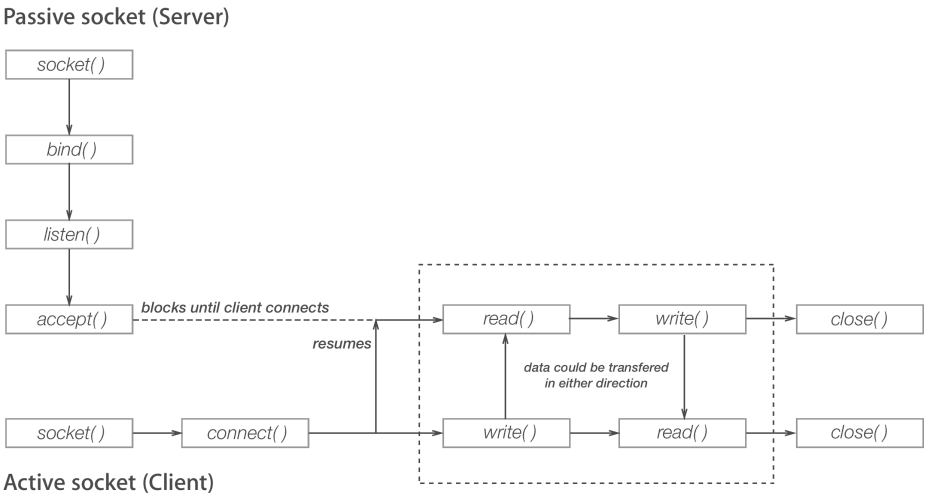
其中， `SocketException` 是我们稍后可能会使用的各种异常， `SocketType` 则表示主动和被动类型的 `Socket` 对象。

其次，我们来定义 `Socket` ：

```
class Socket {
    var socketFd: CInt = -1
    var sockAddrUn: sockaddr_un = sockaddr_un()

    init(socketFilePath: String, type: SocketType) throws {
        // What do we need to code here?
    }
}
```

其中， `socketFd` 表示创建的socket文件句柄， `sockAddrUn` 表示UNIX domain使用的地址对象。而 `init` 方法的两个参数，之前我们已经说过了。现在，项目就可以通过编译了，但我们应该在 `init` 里写点什么呢？只要对比一下服务器和客户端在使用Socket时候在流程上的差异，就可以确定这个问题了：



可以看到，在创建 `Socket` 对象并初始化 `sockaddr_un` 之后，服务器和客户端的流程就不同了，因此，我们让 `init` 方法，就只完成前两件事。在 `init` 方法里，添加下面的代码：

```
init(socketFilePath: String, type: SocketType) throws {
    #if os(Linux)
        socketFd = Glibc.socket(AF_UNIX, SOCK_STREAM, 0)
    #else
        socketFd = Darwin.socket(AF_UNIX, SOCK_STREAM, 0)
    #endif

    if socketFd == -1 {
        throw SocketException.cannotCreateSocketFile
    }

    if (type == .passive) &&
        (remove(socketFilePath) == -1) &&
        (errno != ENOENT) {
        throw SocketException.socketFileAlreadyExists
    }

    try initSockAddr(filePath: socketFilePath)
}
```

在上面这段代码里，我们根据当前所在的平台，分别调用了 `Glibc.socket` 和 `Darwin.socket` ，它们的用法是一样的，都是在调用系统API函数，并指定了Socket的类型。然后，如果返回-1，表示创建失败，我们抛出对应的异常。接下来，如果创建的是服务端Socket，我们就检查下当前是否已经存在同名的Socket文件，如果有就先删掉它，如果删除失败，我们也抛出对应的异常。最后，Socket创建成功后，我们调用 `initSocketAddr` 方法，来设置 `sockAddrUn` 属性。

然后，我们就来实现 `initSockAddr` ：

```
func initSockAddr(filePath: String) throws {
    sockAddrUn.sun_family = (sa_family_t)(AF_UNIX)

    let pathLength = filePath.unicodeScalars.count
    let sockBuffLength =
        Mirror(reflecting: sockAddrUn.sun_path).children.count

    if (pathLength + 1) > sockBuffLength {
        throw SocketException.socketPathTooLong
    }

    memcpy(&sockAddrUn.sun_path, filePath, pathLength)
}
```

它的主要功能就两个：第一个是设置 `sun_family`，这和我们创建 `socket` 的时候，应该是一致的。但当我们要把Socket文件地址写进 `sun_path` 的时候，就不那么容易了。在C里，这是一个 `[char]`，但在Swift里，它是一个有若干 `Int8` 的Tuple，来表示这是一个定长的buffer，我们需要把Socket文件路径中每一个字母的ASCII值写到这个Tuple里。

怎么办呢？一个简单粗暴的办法，就是通过Tuple的metadata获取其中的元素个数，只要路径没有超长，就直接调用 `memcpy` 拷贝内存。至此，在初始化一个 `Socket` 对象的工作，就全部完成了。

在调用 `memcpy` 的时候，有一个细节值得注意，就是Swift会把 `String` 对象自动转换成 `UnsafeRawPointer` 类型，因此，我们在传递第二个参数的时候，并没有使用 & 读取地址。但其它类型则不可以，因此，在传递第一个参数的时候，我们使用了 & 操作符。

What's next?

这样，初始化Socket对象的部分就完成了。现在，稍微休息一下，在下一节，我们完成服务端剩余的部分，绑定、监听 socket 地址，并接受来自客户端的连接请求。

⏮ 关于C中的字符串指针
(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/252)

服务端的Socket demo - II ⏭
(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/254)