Interoperate Swift with C

服务端的Socket demo - I
(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/253)

客户端的Socket demo
(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/255)

服务端的Socket demo - II

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

初始化完 Socket 对象之后，现在，我们又该继续设计API的用法了，按照C里的印象，作为服务端，接下来应该绑定Socket地址，然后在这个地址上监听请求。于是，我们把Swift版本的API设计成这样：

```
do {
    let socket = try Socket(
        socketFilePath: "/tmp/swift_sock_demo")

    try socket.bind()
    try socket.listen(backlog: 10)
}
catch {
    print(error.localizedDescription)
}
```

为了让项目重新恢复编译，我们继续来实现 bind 和 listen(backlog:) 方法。它们的实现基本上都是到C API的转发调用，但 bind() 需要我们处理一些指针类型的差异，来看代码：

```
func bind() throws {
    let rawPointer = UnsafeMutableRawPointer(&sockAddrUn)
    let generalSockAddr = rawPointer.bindMemory(
        to: sockaddr.self,
        capacity: MemoryLayout<sockaddr>.size)

    #if os(Linux)
        let bindResult = Glibc.bind(socketFd, generalSockAddr,
             socklen_t(MemoryLayout<sockaddr_un>.size))
    #else
        let bindResult = Darwin.bind(socketFd, generalSockAddr,
             socklen_t(MemoryLayout<sockaddr_un>.size))
    #endif


    if bindResult == -1 {
        throw SocketException.cannotBindSocketAddress
    }
}
```

这段代码看着有点儿复杂，实际上很简单。在Swift里，系统调用 bind 的第二个参数类型是 UnsafePointer<sockaddr>!，而我们直接传递 sockAddrUn 的地址，得到的类型是 UnsafePointer<sockaddr_un>!，尽管在二进制上 sockaddr 和 sockaddr_un 是完全兼容的，但编译器并不这么想，为了调用 bind，我们必须执行这样的类型转换：

- 先把 UnsafePointer<sockaddr_un> 转换成一个 UnsafeMutableRawPointer；
- 再把 UnsafeMutableRawPointer 转换成一个 UnsafePointer<sockaddr>；

如果你了解了之前我们讲过的指针类型转换规则，理解上面的代码并不会困难。最后，如果 bind 失败了，我们就返回对应的异常。

了解了 bind 的实现之后，我们来实现 listen(backlog:)，它的实现就非常简单了，就是C函数的调用转发，并在错误的时候抛出异常：

```
func listen(backlog: CInt) throws {
    #if os(Linux)
        let result = Glibc.listen(socketFd, backlog)
    #else
        let result = Darwin.listen(socketFd, backlog)
    #endif
    guard result != -1 else {
        throw SocketException.cannotListenOnTheSocketAddress
    }
}
```

这样，我们的项目就又可以通过编译了。

Accept a connection

现在，我们回到*main.swift*，继续设计Socket API，当服务端成功监听之后，我们就要处理来自客户端的连接请求了，在C里，这通常是通过在一个无限循环中反复调用 `accept` 函数完成的。在Swift里，我们希望去掉这个细节，只用一个closure表示收到请求时要执行的动作就好了：

```
do {
    let socket = try Socket(socketFilePath: "/tmp/swift_sock_demo")
    try socket.bind()
    try socket.listen(backlog: 10)

    try socket.accept {
        (fd: CInt) in
        // Handle connection request here
    }
}
catch {
    print(error.localizedDescription)
}
```

由于我们可能需要在closure中读写Socket文件，这里，我们让closure接受一个文件句柄参数。这个参数，由 `Socket.accept` 传递给它。接下来，我们把 `accept` 的closure实现放一放，先来实现 `accept` 方法。继续给 `Socket` 添加下面的代码：

```
func accept(action: (CInt) -> Void) throws {
    while true {
        #if os(Linux)
            let connFd = Glibc.accept(socketFd, nil, nil)
        #else
            let connFd = Darwin.accept(socketFd, nil, nil)
        #endif

        if connFd == -1 {
            throw SocketException.cannotAcceptConnection
        }

        action(connFd)
    }
}
```

基本上，就是C API的一层封装，很简单。在成功 `accept` 之后，我们把新返回的文件句柄发送给 `action` closure。然后，我们回到*main.swift*，来实现接受连接请求之后的代码：

```
try socket.accept {
    var buffer: [CChar] = Array<CChar>(repeating: 0, count: 256)
    var numRead = read($0, &buffer, 256)

    while numRead > 0 {
        if write(STDOUT_FILENO, &buffer, numRead) != numRead {
            fatalError("Partial write...")
        }

        numRead = read($0, &buffer, 256)
    }

    if numRead == -1 {
        fatalError("Read file failed")
    }

    if close($0) == -1 {
        fatalError("Cannot close the connection socket file")
    }
}
```

在上面这段代码里，我们先创建了一个256字节的 `buffer` ，然后，反复调用 `read` 从Socket文件句柄中读取客户端发送的内容。如果读取成功了，我们就把内容写到 `STDOUT_FILENO` ，否则就打印运行时错误后退出。

What's next?

这样，一个简单的UNIX domain的socket服务端就实现完了。纵观整个过程，需要我们特别处理的只有三类内容：

- 如何使用指针类型；
- 如何按字节处理字符串；
- 如何解决跨平台的问题；

在下段视频里，我们基于创建的 `Socket` 类，完成客户端的实现。

⏮ 服务端的Socket demo - I
(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/253)

客户端的Socket demo ⏭
(https://www.boxueio.com/series/interoperate-swift-with-c/ebook/255)

泊学相关

关于泊学 ›

加入泊学 ›

泊学用户隐私以及服务条款 (HTTPS://WWW.BOXUEIO.COM/TERMS-OF-SERVICE)

版权声明 (HTTPS://WWW.BOXUEIO.COM/COPYRIGHT-STATEMENT)

联系泊学

Email: 10[AT]boxue.io (mailto:10@boxue.io)
QQ: 2085489246

友情链接 SwiftV (http://www.swiftv.cn) | Seay信息安全博客 (http://www.cnseay.com) | Swift.gg (http://swift.gg/) | Laravist (http://laravist.com/) | SegmentFault (https://segmentfault.com) | 靛青K的博客 (http://blog.dianqk.org/)