这个事例好处就是本机可以自己测试，一个模拟器就可以。

分两块：1.客户端发送数据和服务器接收 2.服务器发送数据和客户端接收

原理其实是一样的，我这里详细接收了第一种情况，也就是 ‘客户端发送数据和服务器接收’，另一块的原理是一样的，通过调试可以看出运行原理。下面我主要是翻译一下作者提到的问题。

关于简单网络流的介绍

= = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = = =

SimpleNetworkStreams展示了如何使用NSStream API做简单的网络通信

这个样例的目的是非常有限的:它没有完全展示你需要实现一个完全成熟的网络产品(下图),相反,它侧重于如何使用NSStream API将实际的数据应用于整个网络。

SimpleNetworkStreams应该工作在iPhone OS 2.0和以后。 核心网络概念也适用于Mac OS X。

示例包含下列几个重要的类:

SendController[hm]——一个发送文件的视图控制器。

ReceiveController[hm]——一个接收文件的视图控制器。

ReceiveServerController[hm]——一个视图控制器,实现了一个服务器来接收文件。

SendServerController[hm]——一个视图控制器,实现了一个服务器发送文件。

使用示例

- - - - - - - - - - - - - - - - -

您可以测试示例与一个设备(使用环回),一个模拟器(使用环回),两个设备,两个模拟器(在不同的机器上),设备和模拟器,设备和Mac命令行,和一个模拟器和Mac命令行。所有设备和模拟器必须在同一wi - fi网络。

测试的GUI是简单的:

1。在一台设备上运行这个程序(或模拟器)。

2。切换到“接收服务器”选项卡并点击开始。

重要:如果服务器失败开始与消息“注册失败”（netService:(NSNetService \*)sender didNotPublish:(NSDictionary \*)errorDict这个是发布失败）,那么很可能是另一个副本的服务器运行在同一网络。关于这个限制的更多信息,请参阅下面的“简化假设文档”。

3。切换到“发送服务器”选项卡并点击开始

4。如果你使用的是另一个设备或模拟器),运行这个程序在那。

5。切换到“发送”选项卡并利用图像的发送。

6。切换到“接收”选项卡并利用接收接收一个图像。

如果你想要测试对Mac命令行， 打开信息选项卡看看介绍。

它是如何工作的

- - - - - - - - - - - -

这个示例是一个非常简单的应用程序的NSStream API,用于网络。每个视图控制器是一个独立的网络的例子。客户端视图控制器(SendController和ReceiveController)通过硬编码创建一个NSNetService,然后创建一个来自NSOutputStream或NSInputStream的流,然后安排流的异步操作。 大部分的有趣的代码是在-\_startSend / \_startReceive和流:handleEvent:方法。

服务器视图控制器(ReceiveServerController和SendServerController)稍微复杂一些。 他们创建一个监听TCP套接字,包装在一个CFSocket中,然后使用NSNetService Bonjour注册它。当一个连接到达后,他们将在一个NSStream连接的套接字,然后安排,流的异步操作。大部分的有趣的代码是在-\_startServer,\_startSend / \_startReceive,和流:handleEvent:方法。

简化假设

- - - - - - - - - - - - - - - - - - - - - - - -

为了让这个示例容易理解，我做了一些简化假设。如果您正在创建一个真正的应用程序中,您应该阅读“简化假设”的文档，在文档中会看到这些假设可能会影响您的应用程序,以及如何把这个指针示例代码成为符合生产的代码。

学分和版本历史

- - - - - - - - - - - - - - - - - - - - - - - - - - - - -

如果你发现任何问题与此示例,邮件< dts@apple.com >,我将试着修复它们。

1.0(2009年5月)是第一个航运版本。

分享和享受

苹果开发者技术支持

核心操作系统/硬件

2009年5月27日

简化建设

网络通信是困难的。创建一个足够简单的示例,我不得不做一些简化假设。本文档描述了这些假设和提供了你如何将修改这个示例代码用于生产目的一些提示。

代码结构

- - - - - - - - - - - - -

我使代码更加结构化，以使它容易理解,而不是最大化可维护性和灵活性。特别地:

1.网络代码(在某种意义上“模型”)是直接嵌入在控制器代码。

2 有大量的冗余代码的控制器。

在真实的应用程序中,您也许会单独的网络代码到一个“模型”类使用的每个控制器。我不会这么做,因为我希望人们能够很好地理解整个结构通过查看源文件只有一次。

安全

- - - - - - -

这个示例不注意安全问题,这是完全不现实的。当你创建生产网络软件你要不得不担心安全问题。有许多潜在的问题,包括:

1 认证,身份验证是决定你如何确定另一端的网络连接的关键。在许多情况下,但并非总是,身份验证是紧密地绑定到授权(稍后讨论)。大多数人明白,一个网络服务器必须验证其客户;但网络客户端应该验证它的服务器的话则不太明显。没有这种相互身份验证,您可能会泄露敏感的用户数据到一个欺骗服务器。

2 授权——授权是你如何决定是否一个特定的实体被允许做哪些事情。例如,一个简单的照片共享服务器可能会允许所有用户下载,但是只允许一些用户上传。

iPhone开发人员来说一个常见的方法是通过配对来实现身份验证和授权。用户必须采取一些步骤来对两台电脑进行配对,之后他们可以通信,而无需进一步的用户交互。

虽然你可以用不同的API（如NSStream的TLS(见下文)支持和各种安全框架api）来实现配对，但是iphone SDK并没有直接提供配对的支持

3隐私——你必须假定恶意用户正在监视着你的每一块通过网络传输的数据。如果你想以最少的方式传输个人数据,您必须确保这个数据是加密的。

一般来说，我的建议是你应该把用户的数据都要考虑成是敏感的和隐私。因为有可能你觉得这不是隐私但是对用户来说这是。

你保护隐私的主要途径是通过TLS(传输层安全,即SSL或安全套接字层) 这是NSStream直接支持的,所以你会很容易在这个案例中加上加密代码。最大的问题是，为什么这个事例没有使用TLS呢，是身份管理的问题。

安全框架为身份管理提供了一些api，但是通过这个简单的p2p的网络通信事例还不足以说明他们

4恶意攻击—当你在公共网络中交流，你都有可能遭到恶意攻击，在最坏的场景中,攻击者可以创建一个包,使您的应用程序运行任意代码,此时攻击者可以接管正在运行的机器,把它变成一个肉机。你必须仔细检查所有的从网络收到的数据,来防止这种情况的发生。

一个特定形式的恶意攻击是拒绝服务。这个示例并没有试图保护自己免受‘拒绝服务’的攻击。例如,服务器目前仅支持单个连接并没有实现任何超时,连接,所以攻击者可以只是通过简单地打开到服务器的连接并从不关闭它就可以阻止其他有用工作的进行。

性能

- - - - - - - - - - - -

这个示例是简单性为目的而设计的,而不是性能。如果性能是一个严重的问题,你有工作要做。一些显而易见的东西,有可能提高性能包括:

1提高缓冲区大小——代码目前使用32 KB缓冲区对于发送和接收。这可能是太小。

2缓冲区分配——为了简化代码，我分配我的缓冲区作为一个实例变量(发送)或堆(接收)。这是不理想的,尤其随着缓冲区越来越大。你应该在堆上分配缓冲区。

3文件系统I / o——该代码没有试图优化其使用的文件系统。为高速网络,特别是在Mac上,性能的文件系统I / O性能的重要网络I / O。同样,你会想重叠网络和文件系统I / O来防止泡沫在网络“管道”。

4线程——目前是在主线程上使用异步代码NSStreams。你可以让NSStreams运行在单独的线程，这样

可以减少延迟,从而提高性能。

服务检索

- - - - - - - - - - - - - - - - - -

目前使用一个硬编码来实现Bonjour服务名称。这显然是不符合实际的(这意味一开始着在任何时候您只能在网络中运行一个服务器),但这么做我为的是让你忽略其他的东西而更加专注于NSStream。

在 一个叫WiTap的示例代码（也就是我整理的IOS局域网通信示例）实现了Bonjour搜索，或者你可以通过DNS名字连接到一个可全局访问的服务器（您可以使用CFStreamCreatePairWithSocketToHost来创建这样的流）

< http://developer.apple.com/iphone/library/samplecode/WiTap/index.html # / /苹果ref / doc / uid / DTS40007321 >

多个连接

————————————————————

为了保持简单的服务器组件，示例仅支持单个连接在任何给定的时间。这很可能是为你的需求不足。一个真正的服务通常有一些类型的连接对象的实例化来运行一个特定的连接。对于一个这样的例子,请参阅HTTPServer和HTTPConnection类在CocoaHTTPServer样本。

< http://developer.apple.com/samplecode/CocoaHTTPServer/index.html >

可靠性

- - - - - - - - - - - -

该示例是在可靠的范围之内设计的。然而,这些范围内只适合示例代码,而不是广泛被应用。特别地:

o使用的协议通过这个示例是微不足道的。发送一个文件的时候,我们连接并发送的数据,直到所有的数据被发送。接受一个文件的时候,我们连接,然后接收数据,直到我们得到一个文件尾。这不是被应用于商业代码,因为没有办法验证所有的数据已经收到。商业代码将需要实现一些原始TCP的协议来传达信息。例如,您可以预先考虑一个简单的标题,里面包括被传送文件的元数据。

TCP应该提供可靠的数据。然而,这并不总是发生(特别是在世界范围)。商业代码应该使用某种形式的端到端校验和保证数据完整无损地运到了。

当网络不可用的时候，示例并没有试图警告用户,但是一些iPhone OS应用程序都期望能够做到。例如,如果设备处于飞行模式,脱机网络是不可能的,你应该告诉用户。

IPv6

——

iPhone当前操作系统并支持IPv6（现在支持了）,所以这并不是一个主要关心这个示例。然而,客户端组件在没有修改代码的情况下就会支持IPv6。服务器组件需要做一些小改动来支持IPv6。再一次,CocoaHTTPServer示例显示了这是如何实现的。