用户手册

1.3

（Objective-C 版）

目录

前言 **1**

本章提要 1

欢迎使用 Hprose 2

体例 3

菜单描述 3

屏幕截图 3

代码范例 3

运行结果 3

获取帮助 3

电子文档 3

在线支持 3

联系我们 4

第一章 快速入门 **5**

本章提要 5

安装 Hprose for Objective-C 6

创建命令行的 Hello 客户端 6

创建窗口界面的 Hello 客户端 8

创建 iPhone/iPad 的 Hello 客户端 11

第二章 类型映射 **15**

本章提要 15

基本类型 16

值类型 16

引用类型 16

基本类型的映射 17

序列化类型映射 17

反序列化默认类型映射 17

反序列化有效类型映射 18

容器类型 19

列表类型 19

字典类型 20

对象类型 20

第三章 客户端 **24**

本章提要 24

同步调用 25

通过 invoke 方法进行同步调用 25

简单参数传递 25
可变的参数和结果类型 25
引用参数传递 26
自定义类型的传输 26
通过代理对象进行同步调用 27
定义协议 27
可变的参数和结果类型 28
引用参数传递 28
自定义类型 28
异步调用 29
通过 invoke 方法进行异步调用 29
通过代理接口进行异步调用 30
Block 回调 33
异常处理 33
同步调用异常处理 33
异步调用异常处理 34
超时设置 36
HTTP 参数设置 36
HTTP 标头 36
持久连接 36
调用结果返回模式 36
Serialized 模式 37
Raw 模式 37
RawWithEndTag 模式 37

前言

在开始使用 Hprose 开发应用程序前 ,您需要先了解一些相关信息。本章将为您提供这些信息，并告诉您如何获取更多的帮助。

本章提要

- 欢迎使用 Hprose
- 体例
- 获取帮助
- 联系我们

欢迎使用 Hprose

您还在为 Ajax 跨域问题而头疼吗？

您还在为 WebService 的低效而苦恼吗？

您还在为选择 C/S 还是 B/S 而犹豫不决吗？

您还在为桌面应用向手机网络应用移植而忧虑吗？

您还在为如何进行多语言跨平台的系统集成而烦闷吗？

您还在为传统分布式系统开发的效率低下运行不稳而痛苦吗？

好了，现在您有了 Hprose，上面的一切问题都不再是问题！

Hprose (High Performance Remote Object Service Engine) 是一个商业开源的新型轻量级跨语言跨平台的面向对象的高性能远程动态通讯中间件。它支持众多语言，例如.NET，Java，Delphi，Objective-C，ActionScript，JavaScript，ASP，PHP，Python，Ruby，C++，Perl 等语言，通过 Hprose 可以在这些语言之间实现方便且高效的互通。

Hprose 使您能高效便捷的创建出功能强大的跨语言，跨平台，分布式应用系统。如果您刚接触网络编程，您会发现用 Hprose 来实现分布式系统易学易用。如果您是一位有经验的程序员，您会发现它是一个功能强大的通讯协议和开发包。有了它，您在任何情况下，都能在更短的时间内完成更多的工作。

Hprose 是 PHPRPC 的进化版本 ,它除了拥有 PHPRPC 的各种优点之外 ,它还具有更多特色功能。Hprose 使用更好的方式来表示数据，在更加节省空间的同时，可以表示更多的数据类型，解析效率也更加高效。在数据传输上，Hprose 以更直接的方式来传输数据，不再需要二次编码，可以直接进行流式读写，效率更高。在远程调用过程中，数据直接被还原为目标类型，不再需要类型转换，效率上再次得到提高。Hprose 不仅具有在 HTTP 协议之上工作的版本，以后还会推出直接在 TCP 协议之上工作的版本。Hprose 在易用性方面也有很大的进步，您几乎不需要花什么时间就能立刻掌握它。

Hprose 与其它远程调用商业产品的区别很明显——Hprose 是开源的，您可以在相应的授权下获得源代码，这样您就可以在遇到问题时更快的找到问题并修复它，或者在您无法直接修复的情况下，更准确的将错误描述给我们，由我们来帮您更快的解决它。您还可以将您所修改的更加完美的代码或者由您所增加的某个激动人心的功能反馈给我们，让我们能够更好的来一起完善它。正是因为有这种机制的存在，您在使用该产品时，实际上可能遇到的问题会更少，因为问题可能已经被他人修复了。

Hprose 与其它远程调用开源产品的区别更加明显，Hprose 不仅仅在开发运行效率，易用性，跨平台和跨语言的能力上较其它开源产品有着明显的不可取代的综合优势，Hprose 还可以保证所有语言的实现具有一致性，而不会向其他开源产品那样即使是同一个通讯协议的不同实现都无法保证良好的互通。而且 Hprose 具有完善的商业支持，可以在任何时候为您提供所需的帮助。不会向其它没有商业支持的开源软件那样，当您遇到问题时只能通过阅读天书般的源代码的方式来解决。

Hprose 支持许多种语言，包括您所常用的、不常用的甚至从来不用的语言。您不需要掌握 Hprose 支持的所有语言，您只需要掌握您所使用的语言就可以开始启程了。

本手册中有些内容可能在其它语言版本的手册中也会看到，我们之所以会在不同语言的手册中重复这些内容是因为我们希望您只需要一本手册就可以掌握 Hprose 在这种语言下的使用 ,而不需要同时翻阅几本书才能有一个全面的认识。

接下来我们就可以开始 Hprose 之旅啦，不过在正式开始之前，先让我们对本文档的编排方式以及如何获得更多帮助作一下说明。当然，如果您对下列内容不感兴趣的话，可以直接跳过下面的部分。

体例

菜单描述

当让您选取菜单项时，菜单的名称将显示在最前面，接着是一个箭头，然后是菜单项的名称和快捷键。例如“文件→退出”意思是“选择文件菜单的退出命令”。

屏幕截图

Hprose 是跨平台的，支持多个操作系统下的多个开发环境，因此文档中可能混合有多个系统上的截图。

代码范例

代码范例将被放在细边框的方框中：

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello Hprose!");
    }
}
```

运行结果

运行结果将被放在粗边框的方框中：

```
Hello Hprose!
```

获取帮助

电子文档

您可以从我们的网站 http://www.hprose.com/documents.php 上下载所有的 Hprose 用户手册电子版，这些文档都是 PDF 格式的。

在线支持

我们的技术支持网页为 http://www.hprose.com/support.php。您可以在该页面找到关于技术支持的相关信息。

联系我们

如果您需要直接跟我们取得联系，可以使用下列方法：

公司名称	北京蓝慕威科技有限公司
公司地址	北京市海淀区马连洼东馨园 2-2-101 号
电子邮件	市场及大型项目合作： manager@hprfc.com 产品购买及项目定制： sales@hprfc.com 技术支持： support@hprfc.com
联系电话	+86-010-80680756（周一至周五，北京时间早上 9 点到下午 5 点）

第一章 快速入门

使用 Hprose 制作一个简单的分布式应用程序只需要几分钟的时间 ,本章将用一个简单但完整的实例来带您快速浏览使用 Hprose for Objective-C 进行分布式程序开发的全过程。

本章提要

- 安装 Hprose for Objective-C
- 创建命令行的 Hello 客户端
- 创建窗口界面的 Hello 客户端
- 创建 iPhone/iPad 的 Hello 客户端

安装 Hprose for Objective-C

Hprose for Objective-C 是针对 Objective-C 2.0 版本编写的。

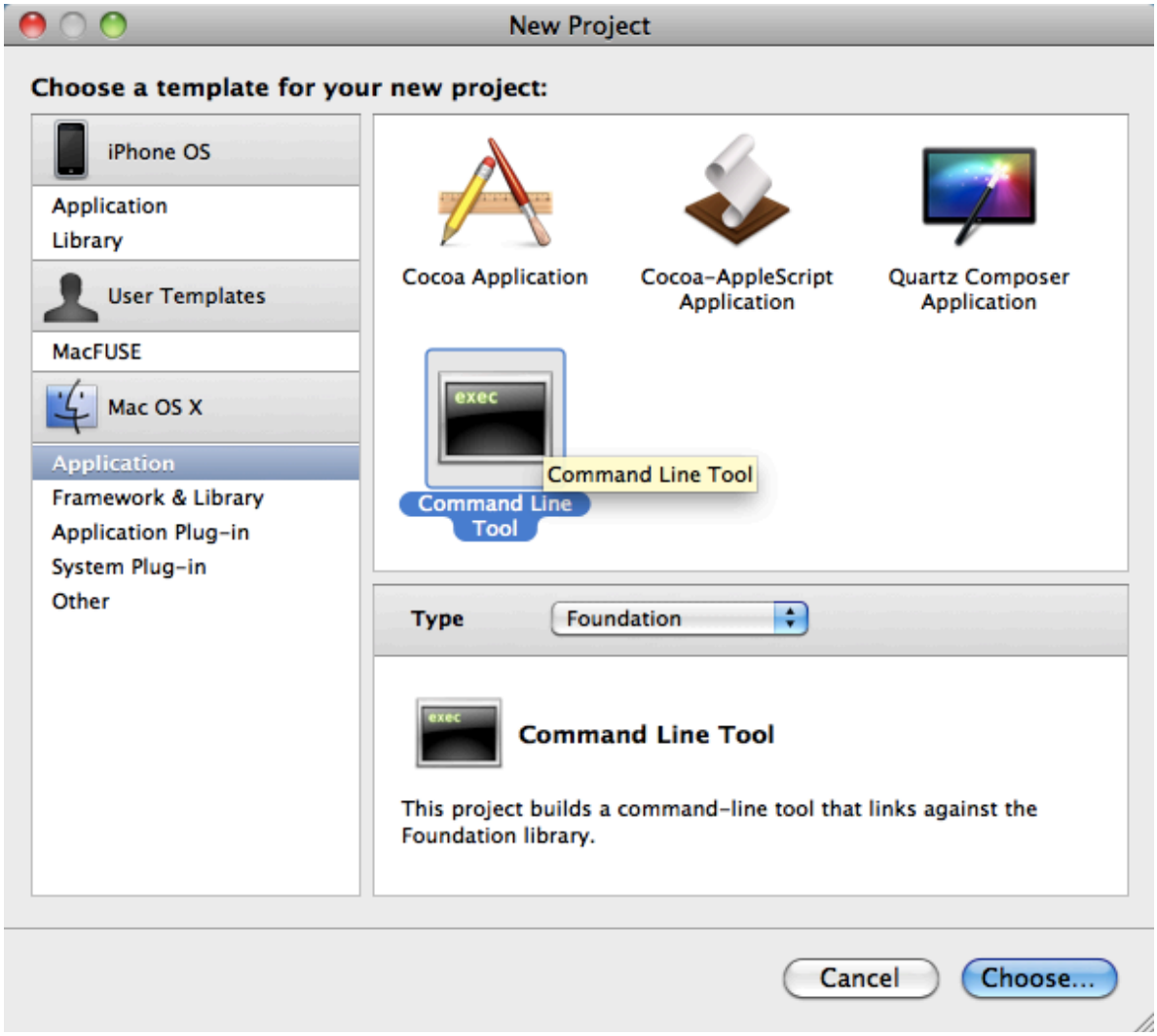
它支持 Mac OS X 10.5 及其更高版本，iPhone、iPod Touch 以及 Apple 最新推出的 iPad。

Hprose for Objective-C 是以源码形式提供的，您可以直接将其源码直接添加到您的项目当中。

创建命令行的 Hello 客户端

Hprose for Objective-C 目前的版本只提供了客户端功能，因此我们还需要一个服务器才能进行讲解和演示。不过这并不是什么问题，Hprose 的客户端和服务器所使用的语言是没有必然联系的，不论服务器是 Java、C#还是 PHP，对客户端来说都是一样的。所以这里我们将使用 PHP 版本的 Hprose 服务器为例来进行讲解。为了方便大家可以直接使用本书中的例子进行练习，我们直接将服务器放在了官方网站上，地址为：http://www.hprose.com/example/，这是一个实际可用的服务，后面所有的例子，都以这个地址所提供的服务为例来进行说明。

在命令行中使用 Hprose 非常简单。打开 XCode，选择“File→New Project…”,然后选择 Command Line Tool，Type 选择 Foundation。然后点击 Choose 按钮。



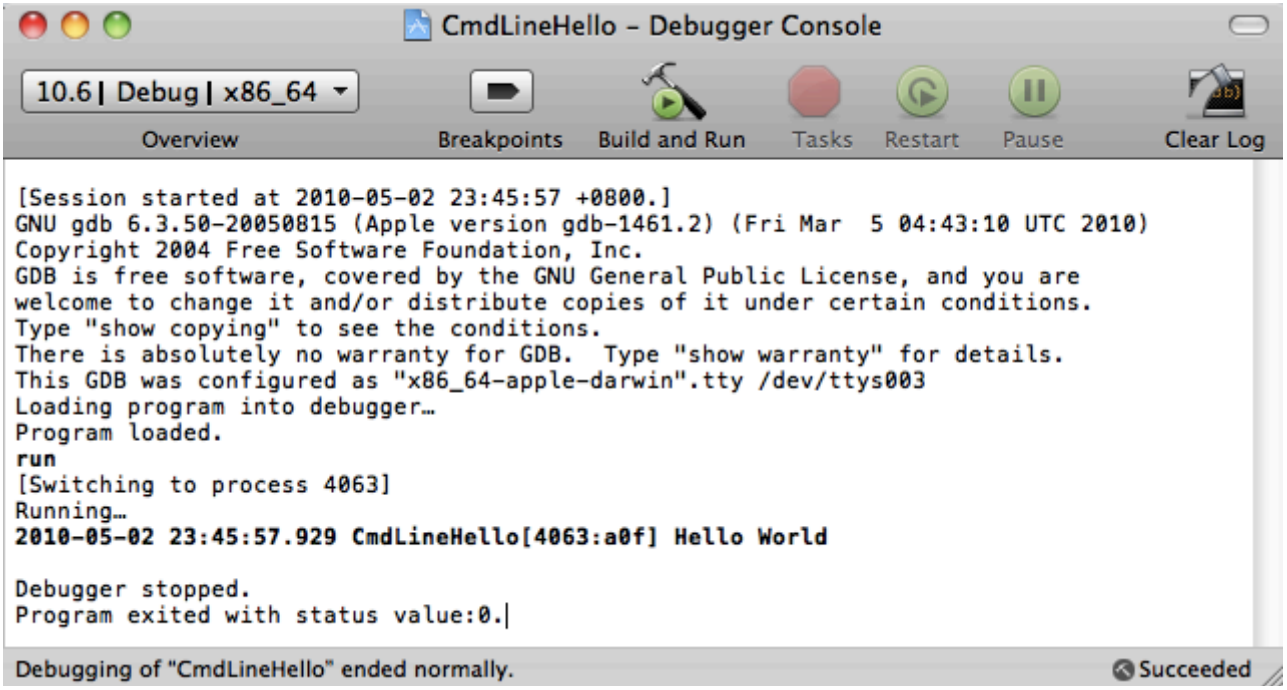
之后在弹出的 Save As 窗口中键入您的项目名称，例如 CmdLineHello。然后点击 Save 按钮。

然后将 Hprose 的全部源文件添加到 Source 之下。或者将已经编译好了静态链接库添加到您的项目中，并把 inlcude 文件也添加到您的项目中。免费的客户端版本只提供静态链接库方式支持。

接下来，我们打开 XCode 刚刚创建的 CmdLineHello.m 文件，将其代码修改为如下内容：

```
#import <Foundation/Foundation.h>
#import "Hprose.h"

@protocol Hello
- (NSString *)hello:(NSString *)name;
@end

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    id<Hello> helloClient = [client useService:@protocol(Hello)];
    NSLog(@"%@", [helloClient hello:@"World"]);
    [pool drain];
    return 0;
}
```

接下来打开“Run→Console”，点击 Build and Run 按钮。您将会看到如下画面：

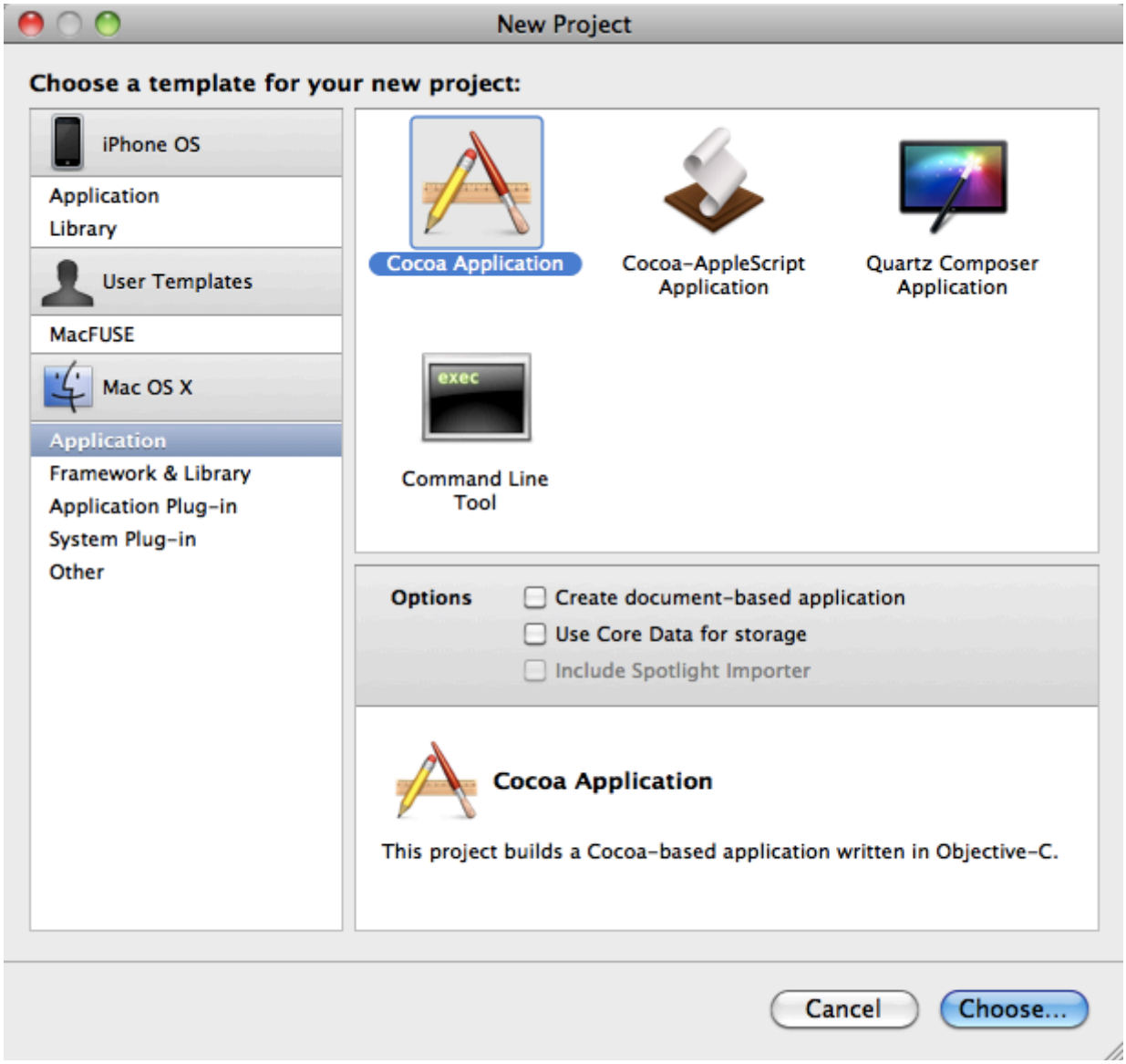


就说明已经成功了。

接下来我们来看看在图形界面下怎样使用 Hprose 来进行远程调用。

创建窗口界面的 Hello 客户端

打开 XCode，选择“File→New Project...”，然后选择 Cocoa Application，Options 不作任何选择。直接点击 Choose 按钮。



之后在弹出的 Save As 窗口中键入您的项目名称，例如 CocoaAppHello。然后点击 Save 按钮。

然后将 Hprose 按前面介绍的方式添加到您的项目中。

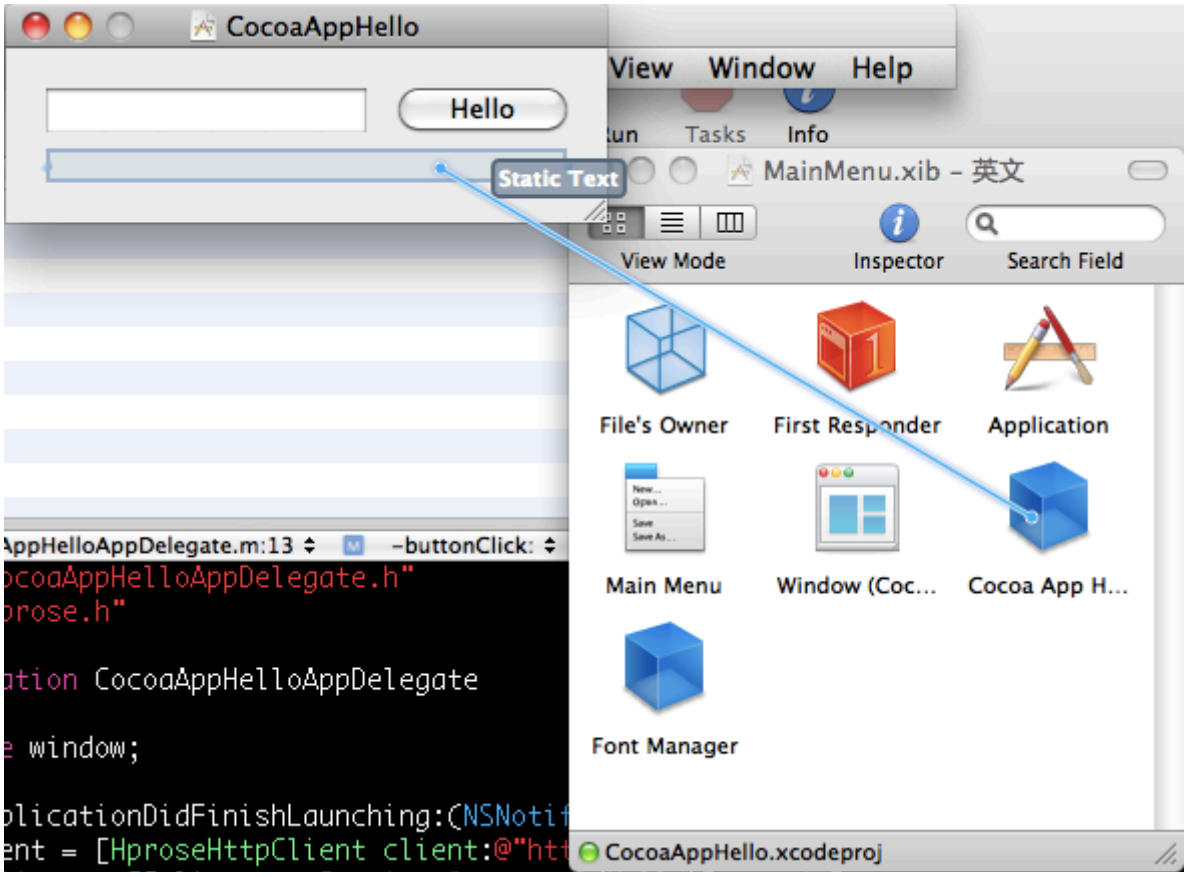
接下来修改 CocoaAppHelloAppDelegate.h，内容如下：

```
#import <Cocoa/Cocoa.h>

@protocol Hello
-(oneway void) hello:(NSString *)name selector:(SEL)selector delegate:(id)delegate;
@end
```

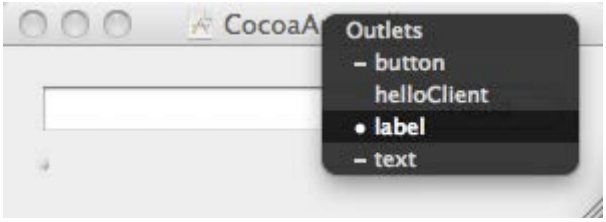
```
@interface CocoaAppHelloAppDelegate : NSObject <NSApplicationDelegate> {
    NSWindow *window;
     IBOutlet id text;
     IBOutlet id button;
     IBOutlet id label;
    id <Hello> helloClient;
}
@property (assign) IBOutlet NSWindow *window;
-(IBAction) buttonClick:(id)sender;
-(void) helloCallback:(NSString *)result;
@end
```

打开 Resources 分组下的 MainMenu.xib 文件，在 Interface Builder 下编辑界面。首先从 Library 中，分别拖放文本框、按钮和文本标签到 CocoaAppHello 窗口中，然后将按钮 Title 改为 Hello，并将文本标签拉伸到适当大小，并将其 Title 设置为空。

在 MainMenu.xib 窗口中，选择 Cocoa App Hello App Delegate，然后右键按住它后拖动至文本标签：

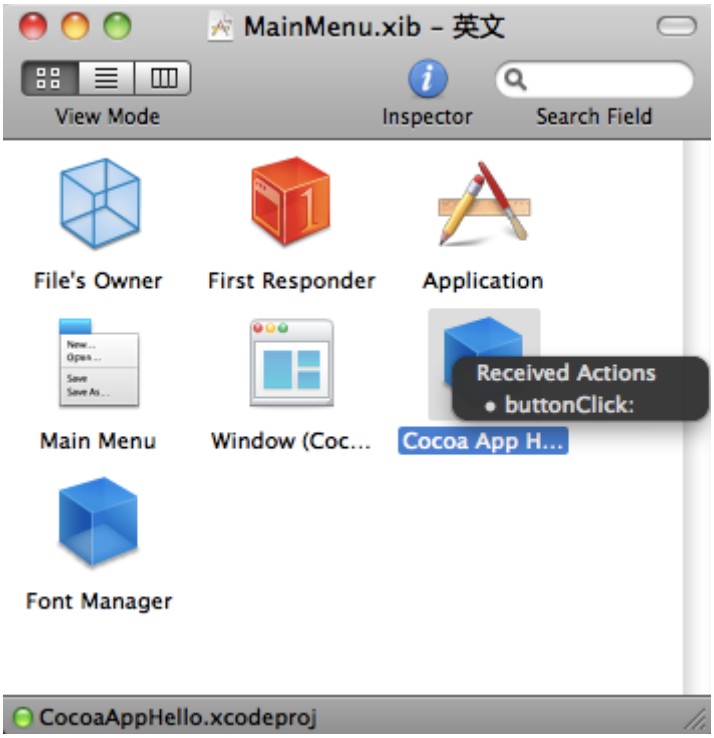


之后松开右键，在弹出的黑色窗口中选择 label。



然后用同样方法，将 button 跟按钮关联，将文本框跟 text 关联。

接下来在 Hello 按钮上按住右键，拖动至 Cocoa App Hello App Delegate 后松开，选择 buttonClick:。

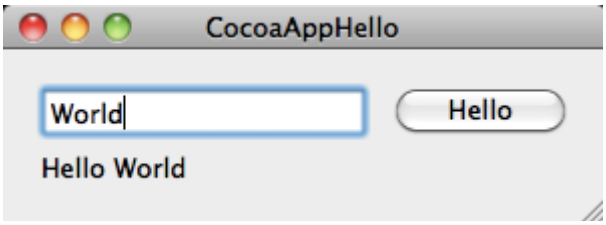


到这里界面就编辑完了，现在保存界面并关闭 Interface Builder。

接下来修改 CocoaAppHelloAppDelegate.m，内容如下：

```
#import "CocoaAppHelloAppDelegate.h"
#import "Hprose.h"

@implementation CocoaAppHelloAppDelegate

@synthesize window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    helloClient = [[client useService:@protocol(Hello)] retain];
}
-(IBAction) buttonClick:(id)sender {
    [helloClient hello:[text stringValue] selector:@selector(helloCallback:) delegate:self]
;
}
-(void) helloCallback:(NSString *)result {
    [label setStringValue: result];
}
@end
```

然后，保存并运行。在运行后的程序中的文本框中输入 World，然后点击 Hello 按钮：

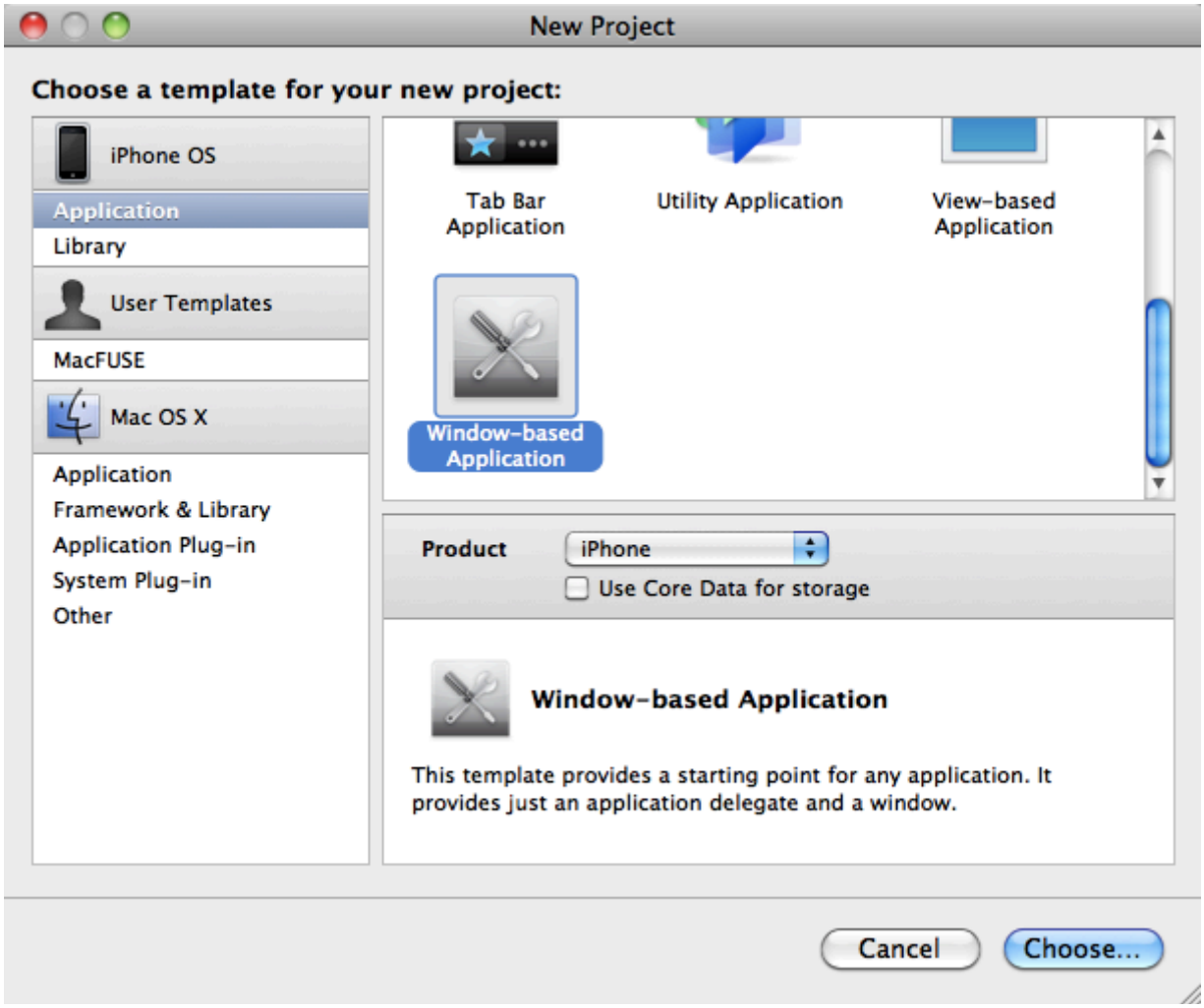


如果您看到上面的画面，就说明客户端创建成功了！

创建 iPhone/iPad 的 Hello 客户端

创建 iPhone/iPad 的 Hprose 客户端跟在 Mac OS X 中创建图形界面客户端方法几乎完全相同。不同之处仅仅是界面操作的语句稍有不同，但这跟 Hprose 本身没有关系。跟 Hprose 有关的代码是完全一致的。但尽管如此，我们还是来看一看如何创建一个 iPhone 客户端。

打开 XCode，选择“File→New Project...”，然后选择“iPhone OS→Window-based Application”，您当然也可以选择其他类型的 iPhone 应用程序，这里选择 Window-based Application 仅仅是为了演示简单。Product 可以任选，然后点击 Choose 按钮。



之后在弹出的 Save As 窗口中键入您的项目名称，例如 iPhoneAppHello。然后点击 Save 按钮。

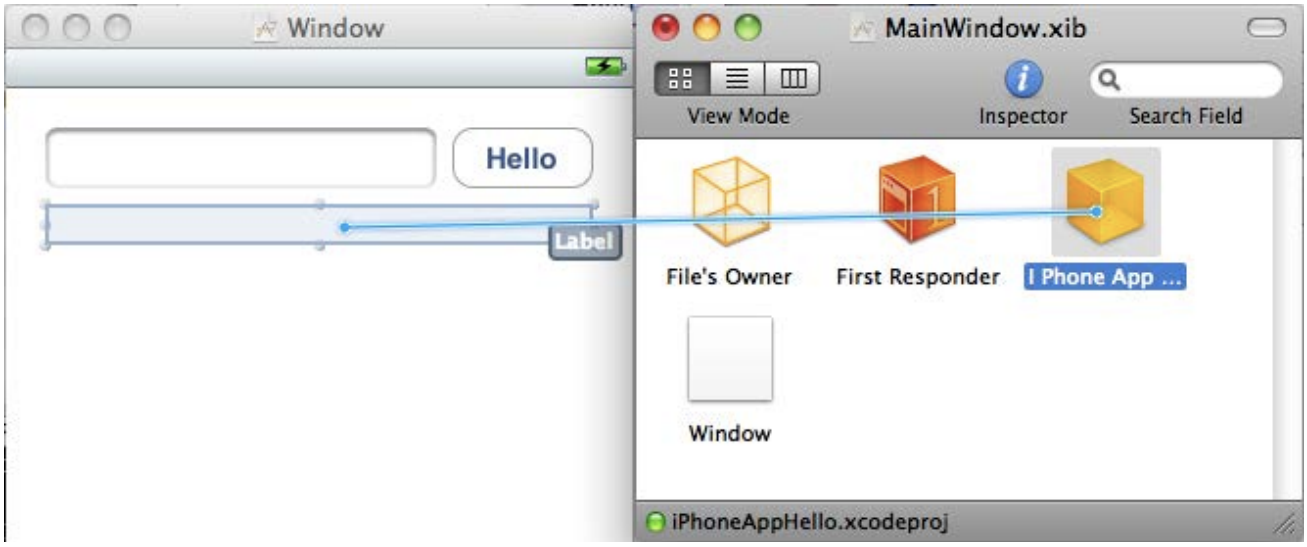
然后将 Hprose 的全部源文件添加到 Source 之下。

接下来修改 iPhoneAppHelloAppDelegate.h，内容如下：

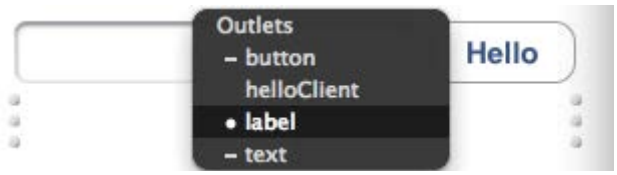
```
#import <UIKit/UIKit.h>

@protocol Hello
-(oneway void) hello:(NSString *)name selector:(SEL)selector delegate:(id)delegate;
@end

@interface iPhoneAppHelloAppDelegate : NSObject <UIApplicationDelegate> {
    UIWindow *window;
    IBOutlet id text;
    IBOutlet id button;
    IBOutlet id label;
    id helloClient;
}
@property (nonatomic, retain) IBOutlet UIWindow *window;
-(IBAction) buttonClick:(id)sender;
-(void) helloCallback:(NSString *)result;
@end
```

打开 Resources 分组下的 MainMenu.xib 文件，在 Interface Builder 下编辑界面。首先从 Library 中，分别拖放文本框、按钮和文本标签到 Window 窗口中，然后将按钮 Title 改为 Hello，并将文本标签拉伸到适当大小，并将其 Title 设置为空。

在 MainMenu.xib 窗口中 ,选择 I Phone App Hello App Delegate ,然后右键按住它后拖动至文本标签：

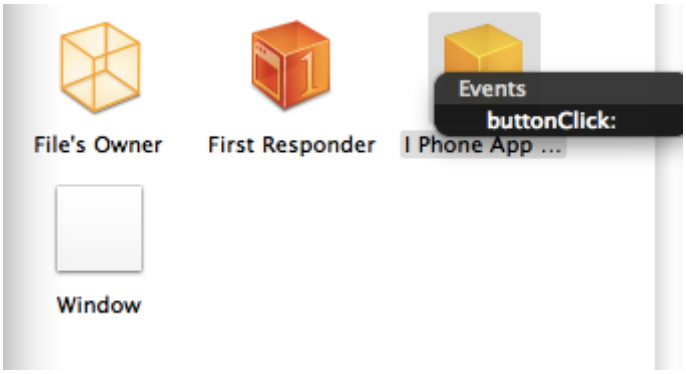


之后松开右键，在弹出的黑色窗口中选择 label。



然后用同样方法，将 button 跟按钮关联，将文本框跟 text 关联。

接下来在 Hello 按钮上按住右键 ,拖动至 I Phone App Hello App Delegate 后松开 ,选择 buttonClick:。



接下来修改 iPhoneAppHelloAppDelegate.m，内容如下：

```
#import "iPhoneAppHelloAppDelegate.h"
#import "Hprose.h"

@implementation iPhoneAppHelloAppDelegate

@synthesize window;

- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictiona
ry *)launchOptions {
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    helloClient = [[client useService:@protocol(Hello)] retain];
    [window makeKeyAndVisible];

     return YES;
}

- (void)dealloc {
    [window release];
    [helloClient release];
    [super dealloc];
}

-(IBAction) buttonClick:(id)sender {
    [helloClient hello:[text text] selector:@selector(helloCallback:) delegate:self];
}

-(void) helloCallback:(NSString *)result {
    [label setText:result];
}

@end
```

然后，保存并运行。在运行后的程序中的文本框中输入 World，然后点击 Hello 按钮：

如果您看到上面的画面，就说明 iPhone 客户端创建成功了！

虽然上面介绍的篇幅比较长，但大部分操作都是跟 Hprose 无关的界面操作。但您或许还有很多疑问，没关系，接下来，就让我们一起对 Hprose for Objective-C 进行深层探秘吧。

第二章 类型映射

类型映射是 Hprose 的基础，正是因为 Hprose 设计有良好的类型映射机制，才使得多语言互通得以实现。本章将对 Hprose for Objective-C 的类型映射进行一个详细的介绍。

本章提要

- 基本类型
- 容器类型
- 对象类型

基本类型

值类型

类型	描述
整型	Hprose 中的整型为 32 位有符号整型数，表示范围是-2147483648 ~ 2147483647（$-2^{31} \sim 2^{31}-1$）。
长整型	Hprose 中的长整型为有符号无限长整型数，表示范围仅跟内存容量有关。
浮点型	Hprose 中的浮点型为双精度浮点型数。
非数	Hprose 中的非数表示浮点型数中的非数（NaN）。
无穷大	Hprose 中的无穷大表示浮点型数中的正负无穷大数。
布尔型	Hprose 中的布尔型只有真假两个值。
字符	Hprose 中的 UTF8 编码的字符，仅支持单字元字符。
空	Hprose 中的空表示引用类型的值为空（null）。
空串	Hprose 中的空串表示空字符串或零长度的二进制型。

其中非数和无穷大其实是特殊的浮点型数据，只不过在 Hprose 中它们有单独的表示方式，这样可以使它们占用更少的存储空间，并得到更快的解析。

另一个可能会引起您注意的是，这里把空和空串也作为值类型对待了。这里把它列为值类型而不是引用类型，是因为 Hprose 中的值类型和引用类型的概念与程序设计语言中的概念不完全相同。这里的值类型是表示在 Hprose 序列化过程中，不做引用计数的类型。在序列化过程中，当遇到相等的值类型时，后写入的值将与先写入的值保持相同的形式，而不是以引用的形式写入。

引用类型

类型	描述
二进制型	Hprose 中的二进制型表示二进制数据，例如字节数组或二进制字符串。
字符串型	Hprose 中的字符串型表示 Unicode 字符串数据，以标准 UTF-8 编码存储。
日期型	Hprose 中的日期型表示年、月、日，年份范围是 0 ~ 9999。
时间型	Hprose 中的时间型表示时、分、秒（毫秒，微秒，毫微秒为可选部分）。
日期时间型	Hprose 中的日期时间型表示某天的某个时刻，可表示本地或 UTC 时间。

空字符串和零长度的二进制型并不总是表示为空串类型，在某些情况下它们也表示为各自的引用类型。

空串类型只是对二进制型和字符串型的特殊情况的一种优化表示。

引用类型在 Hprose 中有引用计数，在序列化过程中，当遇到相等的引用类型时，后写入的值是先前写入的值的引用编号。后面介绍的容器类型和对象类型也都属于引用类型。

基本类型的映射

Objective-C 类型与 Hprose 类型的映射关系不是一一对应的。在序列化和反序列化过程中可能会有一种 Objective-C 类型对应多种 Hprose 类型的情况出现（当然条件会有不同）。我们下面以列表的形式来说明。

序列化类型映射

Objective-C 类型	Hprose 类型
int8_t ,uint8_t ,int16_t ,uint16_t ,int32_t ,char ,unsigned char ,short ,unsigned short，int，long(i386)，以及 NSNumber 中的上述类型	整型
uint32_t，int64_t，uint64_t，long(x86_64)，unsigned int，unsigned long，long long, unsigned long long，以及 NSNumber 中的上述类型，纯数字的 NSString	长整型
float，double，以及 NSNumber 中的上述类型	浮点型
NaN（isnan 为 true）	非数
Infinity（isinf 为 true 且 signbit 为 false）	正无穷大
-Infinity（isinf 为 true 且 signbit 为 true）	负无穷大
true、YES	布尔真
false、NO	布尔假
nil，[NSNull null]	空
单字符字符串	字符
NSData	二进制型
NSString，char *，const char *	字符串型
NSDate	日期/时间/日期时间型

注意，NSDate 本身并不区别 UTC 时间和本地时间。所以默认序列化时使用默认时区。若要以 UTC 时间序列化，只需要将序列化 UTC 参数设置为 YES 即可。

反序列化默认类型映射

默认类型是指在对 Hprose 数据反序列化时，在不指定类型信息的情况下得到的反序列化结果类型。

Hprose 类型	Objective-C 类型

Hprose 类型	Objective-C 类型
整型	NSNumber 中的 intValue
长整型	NSString
浮点型	NSNumber 中的 doubleValue
非数	NSNumber 中的 doubleValue（NaN，isnan 为 true）
正无穷大	NSNumber 中的 doubleValue（Infinity，isinf 为 true 且 signbit 为 false）
负无穷大	NSNumber 中的 doubleValue（-Infinity，isinf 为 true 且 signbit 为 true）
布尔真	NSNumber 中的 boolValue 的 YES
布尔假	NSNumber 中的 boolValue 的 NO
空	nil（作为 NSArray、NSDictionary 元素时为[NSNull null]）
空串	@""
二进制型	NSData
字符型	NSString
字符串型	NSString
日期/时间/日期时间型	NSDate

反序列化有效类型映射

有效类型是指在对 Hprose 数据反序列化时，可以指定的反序列化结果类型。当指定的类型为安全类型时，反序列化总是可以得到结果。当指定的类型为非安全类型时，只有当数据符合一定条件时，反序列化才能得到结果，不符合条件的情况下，可能会得到丢失精度的结果或者抛出异常。当指定的类型为非有效类型时，反序列化时会抛出异常。

Hprose 类型	Objective-C 类型（安全）	Objective-C 类型（非安全）
整型	NSNumber（intValue，longValue，longlongValue，floatValue，doubleValue），NSString	NSNumber（charValue，unsignedCharValue，shortValue，unsignedShortValue，unsignedIntValue，unsignedLongValue，unsignedLongLongValue，boolValue），NSDate
长整型	NSString	NSNumber，NSDate

Hprose 类型	Objective-C 类型（安全）	Objective-C 类型（非安全）
浮点型	NSNumber（doubleValue），NSString	NSNumber(doubleValue 以外的值)，NSDate
非数	NSNumber（floatValue，doubleValue），NSString	无
正无穷大	NSNumber（floatValue，doubleValue），NSString	无
负无穷大	NSNumber（floatValue，doubleValue），NSString	无
布尔真	NSNumber，NSString	无
布尔假	NSNumber，NSString	无
空	NSNumber，NSNull，nil	无
空串	NSString，NSData，NSNumber，NSNull，nil	无
二进制型	NSData	NSString(NSISOLatin1StringEncoding)
字符/字符串型	NSString，NSData	NSNumber
日期/时间/日期时间型	NSDate	无

NSString 和 NSMutableString 在序列化时相同，反序列化时根据指定的类型进行反序列化。NSData 和 NSMutableData 亦然。

容器类型

Hprose 中的容器类型包括列表类型和字典类型两种。

列表类型

Objective-C 中的 NSArray 类型数据被映射为 Hprose 列表类型。例如：

```
NSArray *array = [NSArray arrayWithObjects:@"one", @"two", @"three", nil];
```

Hprose 的列表类型支持对自身的引用，因此，下面的这个 NSMutableArray 对象是可以被正确传递的：

```
NSMutableArray *array = [NSMutableArray array];
[array addObject:array];
```

因为 NSMutableArray 是 NSArray 的子类，所以在序列化时，它们没有区别。在反序列化时，全部按照 NSMutableArray 进行反序列化。

字典类型

Objective-C 中的 NSDictionary 类型数据被映射为 Hprose 字典类型。例如：

```
   NSDictionary *dict = [NSDictionary dictionaryWithObjectsAndKeys:@"Tom", @"name", [NSNum
ber numberWithInt:30], @"age", nil];
```

Hprose 的字典类型也支持对自身的引用，因此，下面的这个 NSMutableDictionary 对象也是可以被正确传递的：

```
   NSMutableDictionary *dict = [NSMutableDictionary dictionary];
   [dict setObject:@"Tom" forKey:@"name"];
   [dict setObject:[NSNumber numberWithInt:30] forKey:@"age"];
   [dict setObject:dict forKey:@"self"];
```

因为 NSMutableDictionary 是 NSDictionary 的子类，所以在序列化时，它们没有区别。在反序列化时，全部按照 NSMutableDictionary 进行反序列化。

有时候我们更希望以强类型方式来传递对象，而不是以字典方式，那么下面我们就来看一下 Hprose 中的对象类型。

对象类型

Objective-C 中自定义类型的对象实例在序列化时被映射为 Hprose 对象类型。那么我们怎么样创建自定义类型呢？很简单，例如在 Objective-C 中可以这样创建自定义类型：

```
typedef enum {
   Unknown, Male, Female, InterSex
} Sex;

@interface User : NSObject {
   NSString *name;
   int age;
   NSDate *birthday;
   Sex sex;
   BOOL married;
}
@property (copy) NSString *name;
@property int age;
@property (retain) NSDate *birthday;
@property Sex sex;
@property BOOL married;
@end
```

```
@implementation User
@synthesize name;
@synthesize age;
@synthesize birthday;
@synthesize sex;
@synthesize married;
@end
```

上面这段代码就定义了一个 User 类。可序列化的自定义类的属性必须也是可序列化类型的，这些类型包括 char、unsigned char、short、unsigned short、int、unsigned int、long、unsigned long、long long、unsigned long long、BOOL、bool、enum、char *、const char *以及各种 id 类型。属性可以通过@property 指令来定义，支持字段属性、动态属性、以及通过 getter、setter 来定义的属性，在通过@synthesize 实现时也支持属性的字段别名。但不支持只读属性。

虽然 Hprose 不支持 Objective-C 1.0，但是如果您的对象属性完全没有使用@property 来定义，而是通过 1.0 那样的方法来定义的话，Hprose 同样支持，例如：

```
typedef enum {
   Unknown, Male, Female, InterSex
} Sex;

@interface User : NSObject {
   NSString *name;
   int age;
   NSDate *birthday;
   Sex sex;
   BOOL married;
}

-(NSString *)name;
-(void)setName:(NSString *)value;
-(int)age;
-(void)setAge:(int)value;
-(NSDate *)birthday;
-(void)setBirthday:(NSDate *)value;
-(Sex)sex;
-(void)setSex:(Sex)value;
-(BOOL)married;
-(void)setMarried:(BOOL)value;

@end

@implementation User

-(NSString *)name {
```

```
    return name;
}

-(void)setName:(NSString *)value {
    if (name != value) {
        [name release];
        name = [value copy];
    }
}

-(int)age {
    return age;
}

-(void)setAge:(int)value {
    age = value;
}

-(NSDate *)birthday {
    return birthday;
}

-(void)setBirthday:(NSDate *)value {
    if (birthday != value) {
        [birthday release];
        birthday = [value retain];
    }
}

-(Sex)sex {
    return sex;
}

-(void)setSex:(Sex)value {
    sex = value;
}

-(BOOL)married {
    return married;
}

-(void)setMarried:(BOOL)value {
    married = value;
}
```

```
@end
```

这样定义效果是相同的。但不要把这两种方式混合使用，当混合使用时，只有@property 定义的属性会被序列化。

如果您只是接收服务器端传来的数据，您甚至在客户端不需要定义接收数据的类，该类会在数据接收时自动被动态定义。

因为 Objective-C 不支持名称空间，因此，当它跟支持名称空间的语言进行交互时，通过在类名中使用下滑线来定义与其它带有名称空间的语言对应的类，例如 Objective-C 中定义的 My_NameSpace_ClassName 与 C#中的 My.NameSpace.ClassName 类是相对应的。另外，类名（包括名空间部分）是区分大小写的。

另外您还可以通过 HproseClassManager 的 registerClass 方法来为类注册别名。

例如您有一个 User 的类，希望传递给 C#，但是在 C#中与之对应的类是 My.NameSpace.User，那么可以这样做：

```
HproseClassManager.registerClass([User class], 'My_NameSpace_User');
```

第三章 客户端

前面我们在快速入门一章里学习了如何创建一个简单的 Hprose for Objective-C 客户端 ,在本章中您将深入的了解 Hprose for Objective-C 客户端的更多细节。

本章提要

- 同步调用
- 异步调用
- 异常处理
- 超时设置
- HTTP 参数设置

同步调用

Hprose 客户端在与服务器通讯时，分同步调用和异步调用两种方式。同步调用的概念和用法相对简单一些，但在图形界面程序、iPhone/iPad 程序中基本不会用到，一般仅用于在命令行程序中使用，下面我们先来介绍同步调用方式。

在同步调用方式下，如果服务器执行出错，或者通讯过程中出现问题（例如连接中断，或者调用的服务器不存在等），则客户端会抛出异常。

直接使用 HproseHttpClient 上的 invoke 方法或者采用代理对象方式都可以进行同步调用 ,但通过代对象方式更方便。

在下面的例子中，我们以调用官方网站（地址为：http://www.hprose.com/example/）提供的服务为例来进行说明讲解。

通过 invoke 方法进行同步调用

通过 invoke 方法调用是最直接、最基本的方式，所以我们先来介绍它。下面的代码我们以命令行程序为例，但是为了节省篇幅，突出重点，这里仅列出关键代码。

简单参数传递

下面这个例子跟快速入门中的例子效果一样，但这个是使用 invoke 方法来进行调用的：

```
id client = [HproseHttpClient client:@" http://www.hprose.com/example/"];
NSLog(@"%@", [client invoke:@"Hello" withArgs:[NSMutableArray arrayWithObject:@"World"]]);
```

可变的参数和结果类型

下面我们再来看对 Sum 方法的调用：

```
id client = [HproseHttpClient client:@" http://www.hprose.com/example/"];
NSLog(@"%@", [client invoke:@"sum" withArgs:[NSMutableArray arrayWithObjects:
                                    [NSNumber numberWithInt:1],
                                    [NSNumber numberWithInt:2],
                                    [NSNumber numberWithInt:3],
                                    [NSNumber numberWithInt:4],
                                    [NSNumber numberWithInt:5],
                                    nil]]);
NSLog(@"%@", [client invoke:@"sum" withArgs:[NSMutableArray arrayWithObjects:
                                    [NSNumber numberWithDouble:6.1],
                                    [NSNumber numberWithDouble:7.2],
                                    [NSNumber numberWithDouble:8.3],
                                    nil]]);
```

这个例子中，参数个数和参数类型都是不同的，返回的结果类型也不同。但都可以正常执行。这说明 Hprose 是支持弱类型参数传递的。

下面看运行结果：

```
2010-05-06 23:33:04.831 CmdLine[46466:a0f] 15
2010-05-06 23:33:04.850 CmdLine[46466:a0f] 21.6
```

引用参数传递

下面我们来继续看对 SwapKeyAndValue 方法的调用：

```
   id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
   NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithObject:@"January" forKey
:@"Jan"];
   NSMutableArray *args = [NSMutableArray arrayWithObject:dict];
   NSLog(@"%@", args);
   [client invoke:@"swapKeyAndValue" withArgs:args byRef:YES];
   NSLog(@"%@", args);
```

运行结果：

```
2010-05-06 23:54:04.835 CmdLine[46615:a0f] (
       {
       Jan = January;
   }
)
2010-05-06 23:54:05.481 CmdLine[46615:a0f] (
       {
       January = Jan;
   }
)
```

从上面的调用演示了引用参数传递。在对 swapKeyAndValue 进行调用时，我们设置了引用参数传递，所以调用后，参数 args 的值也发生了变化，但是需要注意的是，原始的 dict 并没有改变，改变的是参数数组 args 当中的值。

自定义类型的传输

最后我们来看一下对 getUserList 方法的调用，假设我们已经定义了 User 类，并且该类定义如前一章中自定义类型的例子一样，那么下面是调用程序：

```
   id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
   NSArray *users = [client invoke:@"getUserList"];
   for (id user in users) {
      NSLog(@"%@, %d, %@, %d, %d",
            [user name],
            [user age],
            [user birthday],
```

```
            [user sex],
            [user married]);
    }
```

运行结果：

```
2010-05-07 00:02:11.846 CmdLine[46665:a0f] Amy, 26, 1983-12-03 00:00:00 +0800, 2, 1
2010-05-07 00:02:11.847 CmdLine[46665:a0f] Bob, 20, 1989-06-12 00:00:00 +0900, 1, 0
2010-05-07 00:02:11.848 CmdLine[46665:a0f] Chris, 29, 1980-03-08 00:00:00 +0830, 0, 1
2010-05-07 00:02:11.852 CmdLine[46665:a0f] Alex, 27, 1992-06-14 00:00:00 +0800, 3, 0
```

那如果我们不定义 User 类，可不可以调用呢？可以，但程序需要稍作修改：

```
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    NSArray *users = [client invoke:@"getUserList"];
    for (id user in users) {
        NSLog(@"%@, %@, %@, %@, %@",
            [user name],
            [user age],
            [user birthday],
            [user sex],
            [user married]);
    }
```

修改后的程序跟修改前唯一的区别就是 NSLog 的格式字符串，也就是说如果客户端没有定义要接受的类，则自动生成的类所有属性都是 id 类型的。当然在编译的过程中，会有警告说 age、birthday 等方法没有找到，但这并不影响执行结果。程序运行结果完全一致。

通过代理对象进行同步调用

看完通过 invoke 进行同步调用的方式后，再来看一下通过接口进行同步调用的方式。

定义协议

为了调用上面的方法，我们需要先定义协议（相当于 C#、Java 的接口），下面是协议的定义：

```
@protocol Exam
- (int)sum:(int)a and:(int)b and:(int)c and:(int)d and:(int)e;
- (double)sum:(double)a and:(double)b and:(double)c;
- (NSDictionary *)swapKeyAndValue:(inout NSMutableDictionary **)dict;
- (id)getUserList;
@end
```

从上面的接口我们可以看出，在 Hprose 中，客户端和服务器端的接口不必完全一致，这就大大增加了灵活性，也更加方便了跟弱类型语言进行交互。

可变的参数和结果类型

下面我们再来看对 sum 方法的调用：

```
id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
id<Exam> exam = [client useService:@protocol(Exam)];
NSLog(@"%d", [exam sum:1 and:2 and:3 and:4 and:5]);
NSLog(@"%f", [exam sum:6.1 and:7.2 and:8.3]);
```

这个程序运行结果跟前面用 invoke 实现的 sum 调用是相同的，这个程序看上去更简洁一些，不过需要事先将协议定义好才可以。用代理对象方式可以直接使用原生类型，而不需要使用 NSNumber 来包装。

引用参数传递

下面继续来看对 swapKeyAndValue 方法的调用：

```
id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
id<Exam> exam = [client useService:@protocol(Exam)];
NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithObject:@"January" forKey
:@"Jan"];
NSLog(@"%@", dict);
[exam swapKeyAndValue:&dict];
NSLog(@"%@", dict);
```

运行结果如下：

```
2010-05-07 10:06:52.276 CmdLine[47463:a0f] {
    Jan = January;
}
2010-05-07 10:06:52.809 CmdLine[47463:a0f] {
    January = Jan;
}
```

当参数在协议声明中被标记为 inout、out 或 ref 时，该参数将被以引用方式传递。注意，这里参数不是 dict 对象本身，而是 dict 的地址，因此最后的结果 dict 对象本身会改变。这一点与通过 invoke 调用时是不同的，采用代理对象方式更直观更方便。

自定义类型

下面来看看 getUserList 怎么调用：

```
id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
id<Exam> exam = [client useService:@protocol(Exam)];
NSArray *users = [exam getUserList];
for (id user in users) {
    NSLog(@"%@, %@, %@, %@, %@",
          [user name],
```

```
            [user age],
            [user birthday],
            [user sex],
            [user married]);
    }
```

运行结果跟直接使用 invoke 方式相同，User 类同样不需要定义，返回结果类型用 id 表示的话，就可以表示所有已定义或未定义类型了。

这个例子很好的说明了 Hprose 使用的易用性和灵活性，不用多做解释相信您也已经看懂了。

异步调用

下面我们来开始另一个重要的话题，那就是异步调用。

异步调用相对于同步调用来说确实要难以掌握一些，但是在很多情况下我们却很需要它。那究竟什么时候我们需要使用异步调用呢？

很多时候我们并不确定在进行远程调用时是否能够立即得到返回结果，因为可能由于带宽问题或者服务器本身需要对此调用进行长时间计算而不能马上返回结果给客户端。这种情况下，如果使用同步远程调用，客户端执行该调用的线程将被阻塞，并且在主线程中执行同步远程调用会造成用户界面冻结，这是用户无法忍受的。这时，我们就需要使用异步调用。

因此我们应该在图形界面应用程序、iPhone、iPad 应用程序中使用异步调用而不应该使用同步调用。

虽然您也可以使用多线程加同步调用来完成异步调用，但您完全不必这样做。因为 Hprose 提供的异步调用更加简单，更加高效。

通过 invoke 方法进行异步调用

通过 invoke 方式进行异步调用跟同步调用差不多，唯一的区别就是异步调用多了一个回调参数。

Hprose 1.0 for Objective-C 提供了 2 种回调方式，一种是函数回调，另一种是委托回调。通常我们在应用中基本上用不到函数回调方式，因为委托回调方式功能更强大，也更易用。所以，下面我们主要委托回调方式为主来介绍异步调用。

委托实际上是一个对象，在设置回调时，您可以通过设置 selector 来指定具体的回调方法，也可以不指定。在不指定 selector 的情况下，Hprose 会在委托对象中按照 callback、callback:、callback:withArgs: 这样的顺序查找回调方法。这三个方法名分别对应没有参数、一个参数和两个参数的回调方法。其中第一个参数表示返回结果，第二个参数表示调用参数。如果您不是在进行引用参数传递的调用，那么第二个参数的参数值，跟您调用时的参数值是一致的。在非引用参数传递调用中，如果您在处理结果的回调方法中并不关心调用时的参数，可以使用只有结果参数的回调方法。如果您连返回结果也不关心（或者调用的远程方法本来就没有返回值），可以使用无参回调方法。通常情况下，指定 selector 更方便灵活一些。

当您通过 invoke 所调用的方法执行完毕时，您所指定的回调方法将会被调用，其中的参将会自动被传入。

关于通过 invoke 进行异步调用，我想不用举太多例子，下面这个简单的例子就可以很好的说明如何来使用了：

```
-(IBAction) buttonClick:(id)sender {
```

```
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    [client setDelegate:self];
    [client invoke:@"Hello"
         withArgs:[NSMutableArray arrayWithObject:[text stringValue]]
         selector:@selector(helloCallback:)];
}

-(void) helloCallback:(NSString *)result {
    [label setStringValue: result];
}
```

HproseHttpClient 的 client 类方法创建的对象是自动释放对象 ,如果在 applicationDidFinishLaunching 或 awakeFromNib 等应用程序初始方法中创建的话，需要用 retain 来保留其引用，否则在后面的方法中将因为其已被释放而不能使用。也可以使用 alloc 和 init 来创建对象，这样也可以保留对象引用。

client 可以直接设置 delegate 属性，该属性将设置好默认的委托对象。后面调用中，在只设定 selector 的情况下，client 会在默认委托对象中查找回调方法。

回调方法的返回值类型在回调方法中通过第一个参数定义即可。

通过代理接口进行异步调用

除了可以通过 invoke 方式外，您也可以通过接口方式来进行异步调用，这里我们来举一个稍微复杂点的例子，来说明引用参数传递，容器类型和自定义类型传输，以及如何在调用中指定要返回的结果类型。

这里以 Cocoa 图形界面应用程序为例，项目创建方式跟快速入门中的方式相同，这里不再重复。

先看定义文件 CocoaAppDelegate.h：

```
#import <Cocoa/Cocoa.h>

@protocol Exam
-(oneway void) swapKeyAndValue:(byref NSDictionary *)dict selector:(SEL)selector delegate:
(id)delegate;
-(oneway void) getUserList:(SEL)selector;
@end

@interface CocoaAppDelegate : NSObject <NSApplicationDelegate> {
    NSWindow *window;
    IBOutlet id button;
    IBOutlet id button2;
    IBOutlet id label;
    id <Exam> exam;
}

@property (assign) IBOutlet NSWindow *window;
-(IBAction) swapKeyAndValueClick:(id)sender;
-(IBAction) getUserListClick:(id)sender;
```

```
-(void) swapKeyAndValueCallback:(NSDictionary *)result withArgs:(NSArray *)args;
-(void) getUserListCallback:(NSArray *)result;

@end
```

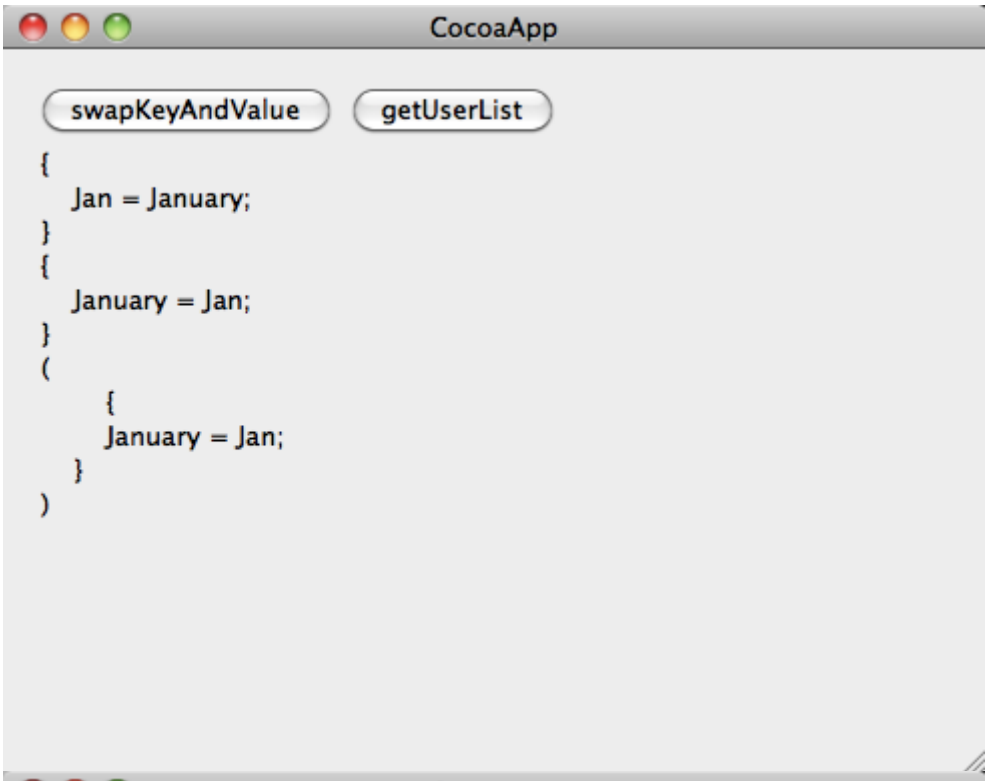
这个协议定义中，swapKeyAndValue 是引用参数传递，并同时指定 selector 和 delegate，而 getUserList 是非引用参数传递，只指定了 selector。注意，如果同时定义 selector 和 delegate，那么 delegate 必须紧随 selector 之后，并且必须是最后一个参数。如果不指定 delegate，那么 selector 必须为最后一个参数。参数前的 selector 字符串任意，不需要必须是 selector:和 delegate:，但是第一个参数前的 selector 字符串（去掉:之后）必须与远程方法名相同。

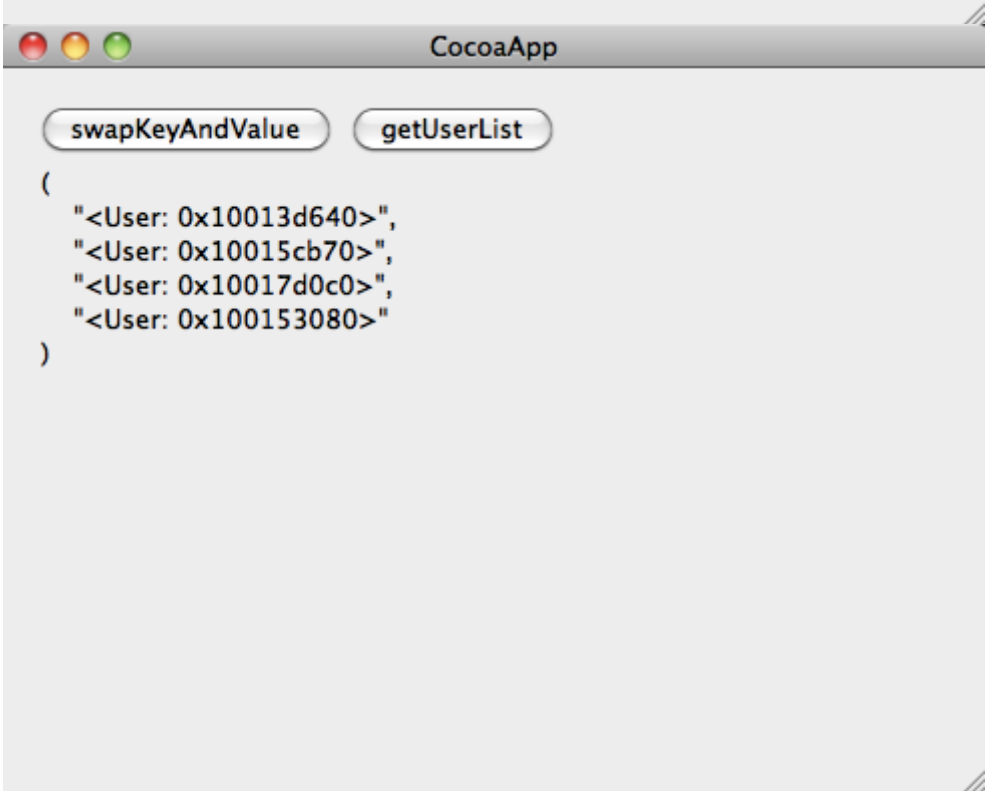
下面看主程序部分。

```
#import "CocoaAppDelegate.h"
#import "Hprose.h"

@implementation CocoaAppDelegate
@synthesize window;

- (void)applicationDidFinishLaunching:(NSNotification *)aNotification {
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    [client setDelegate:self];
    exam = [[client useService:@protocol(Exam)] retain];
}

-(IBAction) swapKeyAndValueClick:(id)sender {
    NSMutableDictionary *dict = [NSMutableDictionary dictionaryWithObject:@"January" forKey
:@"Jan"];
    [label setStringValue:[dict description]];
    [exam swapKeyAndValue:dict selector:@selector(swapKeyAndValueCallback:withArgs:) delega
te:self];
}

-(void) swapKeyAndValueCallback:(NSDictionary *)result withArgs:(NSArray *)args {
    [label setStringValue:[NSString stringWithFormat:@"%@\n%@\n%@",
                           [label stringValue],
                           [result description],
                           [args description]]];
}

-(IBAction) getUserListClick:(id)sender {
    [exam getUserList:@selector(getUserListCallback:)];
}

-(void) getUserListCallback:(NSArray *)result {
    [label setStringValue:[result description]];
```

```
}

@end
```

上面的 label 是一个多行文本控件。程序很简单，下面是运行截图：





在异步调用时，异步方法返回值必须要定义为(oneway void)，oneway 表明了这个是异步调用。参数仍然可以用 byref 来表示引用传递，但这里不用指针做参数，而且参数本身在异步调用后并不改变，改变后

的值体现在回调方法的第二个参数上。

异步调用时，同样可以传递自定义对象类型，在接受自定义对象时，仍然可以接受客户端未定义的类的对象，该对象所在类会自动生成。

Block 回调

Hprose 1.2 for Objective-C 提供了 Block 回调支持。Block 很像一个匿名函数，并且支持闭包特性。关于 Blocks 编程的的更多内容请参见：Blocks Programming Topics，如果您需要在 Mac OS X 10.5 或者 iPhone 环境下使用 Blocks 编程，可以参见 http://code.google.com/p/plblocks/。

不论是使用 invoke 或者代理接口方式，都可以使用 Block 回调，例如在使用代理接口时，可以这样定义接口：

```
@protocol IHello
-(oneway void) hello:(NSString *)name block:(HproseBlock)block;
@end
```

其中，HproseBlock 就是 Block 回调类型，它的定义是这样的：

```
typedef void (^HproseBlock)(id, NSArray *);
```

跟 HproseCallback 的定义很像，只是函数指针变成了 Block。

但是 Block 在调用时更加灵活，例如：

```
-(IBAction) buttonClick:(id)sender {
    [ro hello:[text stringValue] block:^(id result, NSArray *args) {
        [label setStringValue: result];
    }];
}
```

按钮事件可以这样写，而不需要单独定义一个回调函数或者回调方法了。

另外需要注意，block 是在栈上创建的，在 Hprose 1.2 中，你需要调用 copy 方法将它复制到堆上，并调用 autorelease 来自动释放它。否则在回调时会因为 block 已经释放而调用出错。而在 Hprose 1.3 中，这个复制和释放的动作完全由 Hprose 来接管了，因此直接采用上面的写法就可以啦。

异常处理

同步调用异常处理

同步调用下的发生的异常将被直接抛出，使用@try...@catch 语句块即可捕获异常，通常服务器端调用返回的异常是 HproseException 类型。但是在调用过程中也可能抛出其它类型的异常，为了保险，您可以使用@catch 捕获 NSException 类型来处理全部可能发生的异常。例如：

```
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    @try {
```

```
        [client invoke:@"notExist"];
    }
    @catch (NSException * e) {
        NSLog(@"%@", e);
    }
```

运行结果如下：

```
2010-05-08 01:04:04.251 CmdLine[50450:a0f] Can't find this function notexist().
file: /var/www/hprfc/website/example/hproseHttpServer.php
line: 157
trace: #0 /var/www/hprfc/website/example/hproseHttpServer.php(385): HproseHttpServer->doIn
voke()
#1 /var/www/hprfc/website/example/hproseHttpServer.php(402): HproseHttpServer->handle()
#2 /var/www/hprfc/website/example/index.php(60): HproseHttpServer->start()
#3 {main}
```

异步调用异常处理

异步调用时，如果调用过程中发生异常，异常将不会被抛出。但会通过 NSLog 记录异常信息。

例如下面这个程序：

```
@protocol Exam
-(oneway void) notExist:(SEL)selector;
@end
-(IBAction) notExistClick:(id)sender {
    [exam notExist:@selector(notExistCallback)];
}
-(void) notExistCallback {
}
```

调用了一个服务器端不存在的方法 notExist ,这个程序运行后 ,不会有异常抛出 ,但确实有异常发生了。您会在控制台上看到这样的日志信息：

```
2010-05-08 01:56:02.377 CocoaApp[50770:a0f] Can't find this function notexist().
file: /var/www/hprfc/website/example/hproseHttpServer.php
line: 157
trace: #0 /var/www/hprfc/website/example/hproseHttpServer.php(385): HproseHttpServer->doIn
voke()
#1 /var/www/hprfc/website/example/hproseHttpServer.php(402): HproseHttpServer->handle()
#2 /var/www/hprfc/website/example/index.php(60): HproseHttpServer->start()
#3 {main}
```

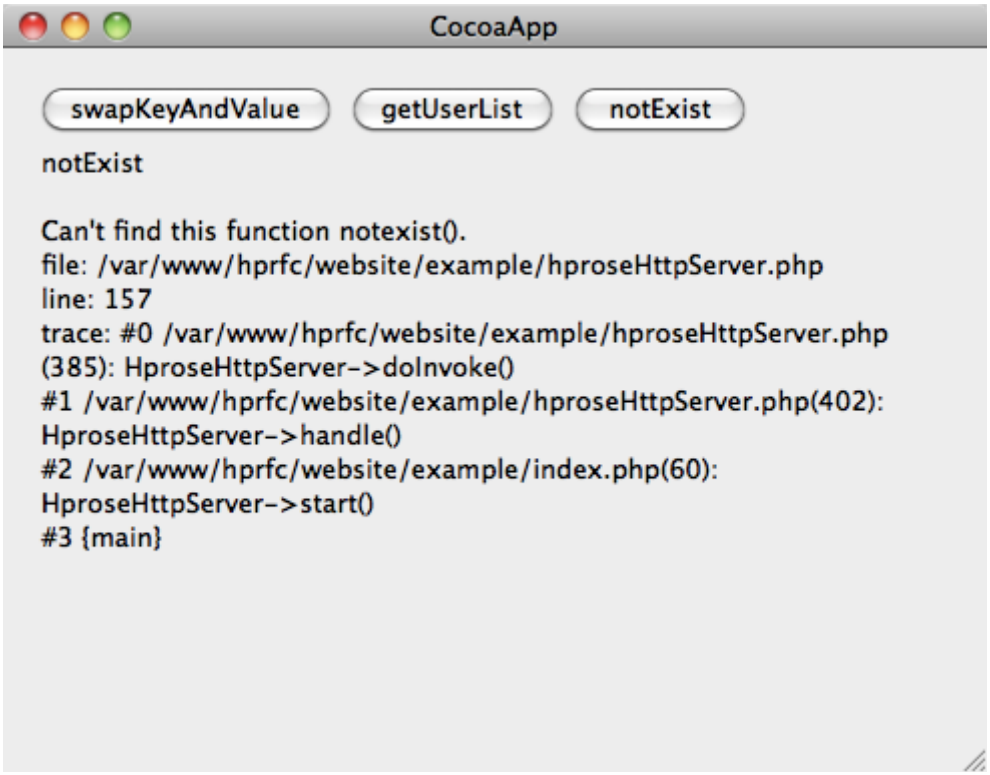
如果您希望替换默认的处理这些异常的方式 ,只需要给 Hprose 客户端对象指定合适的 OnError 事件即可，OnError 事件也是一个回调方法，使用非常简单，例如上面程序只需要在初始化 client 对象时，加上：

```
    id client = [HproseHttpClient client:@"http://www.hprose.com/example/"];
    [client setDelegate:self];
    [client setOnError:@selector(errorHandler:withException:)];
```

并加上 errorHandler:withException 方法实现即可：

```
-(void) errorHandler:(NSString *)name withException:(NSException *)e {
    [label setStringValue:[NSString stringWithFormat:@"%@\n\n%@", name, [e description]]];
}
```

这时运行结果变为



该事件第一个参数为抛出异常的方法名，第二个为抛出的异常。

该事件只对异步调用有效，同步调用下的异常将被直接抛出，而不会被该事件捕获并处理。

Hprose 1.2 for Objective-C 还提供了异步的 Block 错误处理，设置方法是：

```
    [client setErrorHandler:[[^(NSString *name, NSException *e) {
        [label setStringValue:[name stringByAppendingFormat:@": %@", [e description]]];
    } copy] autorelease]];
```

同样，在 Hprose 1.3 中，可以采用下面的写法，而不用再写多余的复制和释放：

```
   [client setErrorHandler:^(NSString *name, NSException *e) {
        [label setStringValue:[name stringByAppendingFormat:@": %@", [e description]]];
    }];
```

超时设置

默认情况下，连接超时时间是 30 秒，您可以通过 timeout 属性来设置连接超时时间，该属性为浮点类型，单位是秒。

HTTP 参数设置

目前的版本只提供了 http 客户端实现，针对于 http 客户端，有一些特别的设置，例如 http 标头、持久连接等设置，下面我们来分别介绍。

HTTP 标头

有时候您可能需要设置特殊的 http 标头，例如当您的服务器需要 Basic 认证的时候，您就需要提供一个 Authorization 标头。设置标头很简单，只需要调用

```
- (void) setValue:(NSString *)value forHTTPHeaderField:(NSString *)field;
```

方法来设置 HTTP 标头即可。该方法的第一个参数为标头值，第二个参数为标头名，这两个参数都是字符串型。如果将第一个参数设置为 nil，则表示删除这个标头。

标头名不可以为以下值：

- Context-Type
- Context-Length
- Host

因为这些标头有特别意义，客户端会自动设定这些值。

另外，Cookie 这个标头不要轻易去设置它，因为设置它会影响 Cookie 的自动处理，如果您的通讯中用到了 Session，通过该方法来设置 Cookie 标头，将会影响 Session 的正常工作。

您也可以使用 header 属性来直接设置 HTTP 标头。

持久连接

默认情况下，持久连接是关闭的。通常情况下，客户端在进行远程调用时，并不需要跟服务器保持持久连接，但如果您有连续的多次调用，可以通过开启这个特性来优化效率。

跟持久连接有关的属性有两个，它们分别是 keepAlive 和 keepAliveTimeout，将 keepAlive 属性设置为 YES 时，表示开启持久连接特征。keepAliveTimeout 表示持久连接超时时间，单位是秒，默认值是 300 秒。

调用结果返回模式

有时候调用的结果需要缓存到文件或者数据库中，或者需要查看返回结果的原始内容。这时，单纯的普通结果返回模式就有些力不从心了。Hprose 1.3 提供更多的结果返回模式，默认的返回模式是 Normal，开发者可以根据自己的需要将结果返回模式设置为 Serialized，Raw 或者 RawWithEndTag。

Serialized 模式

Serialized 模式下，结果以序列化模式返回，在 Objective-C 中，序列化的结果以 NSData *类型返回。因为该模式并不对结果直接反序列化，因此返回速度比普通模式更快。

在调用时，通过在回调方法参数之后，增加一个结果返回模式参数来设置结果的返回模式，结果返回模式是一个枚举值，它的有效值在 HproseResultMode 枚举中定义。它们分别是：

- HproseResultMode_Normal
- HproseResultMode_Serialized
- HproseResultMode_Raw
- HproseResultMode_RawWithEndTag

Raw 模式

Raw 模式下，返回结果的全部信息都以序列化模式返回，包括引用参数传递返回的参数列表，或者服务器端返回的出错信息。该模式比 Serialized 模式更快。

RawWithEndTag 模式

完整的 Hprose 调用结果的原始内容中包含一个结束符，Raw 模式下返回的结果不包含该结束符，而 RawWithEndTag 模式下，则包含该结束符。该模式是速度最快的。

这三种模式主要用于实现存储转发式的 Hprose 代理服务器时使用，可以有效提高 Hprose 代理服务器的运行效率。