用户手册

1.3

(ActionScript 版)

目录

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 ActionScript 进行分布式程序开发的全过程。

本章提要

- 安装 Hprose for ActionScript
- 在 Flash 中创建 Hprose 的 Hello 客户端
- 在 Flex 中创建 Hprose 的 Hello 客户端
- 在 HaXe 中创建 Hprose 的 Hello 客户端

安装 Hprose for ActionScript

Hprose for ActionScript 分为 ActionScript 2.0 和 ActionScript 3.0 两个版本。

ActionScript 2.0 版本支持 Flash 7 及其更高版本的 Flash 应用开发。

ActionScript 2.0 版本支持 Flash Lite 2.0 及其更高版本的 Flash Lite 应用开发。

ActionScript 3.0 版本支持 Flash 9 及其更高版本的 Flash 应用开发。

ActionScript 3.0 版本支持 Flash Lite 4.0 及其更高版本的 Flash Lite 应用开发。

ActionScript 3.0 版本支持 Flex 应用开发。

ActionScript 3.0 版本支持 AIR 应用开发。

ActionScript 3.0 版本支持 Haxe 应用开发。

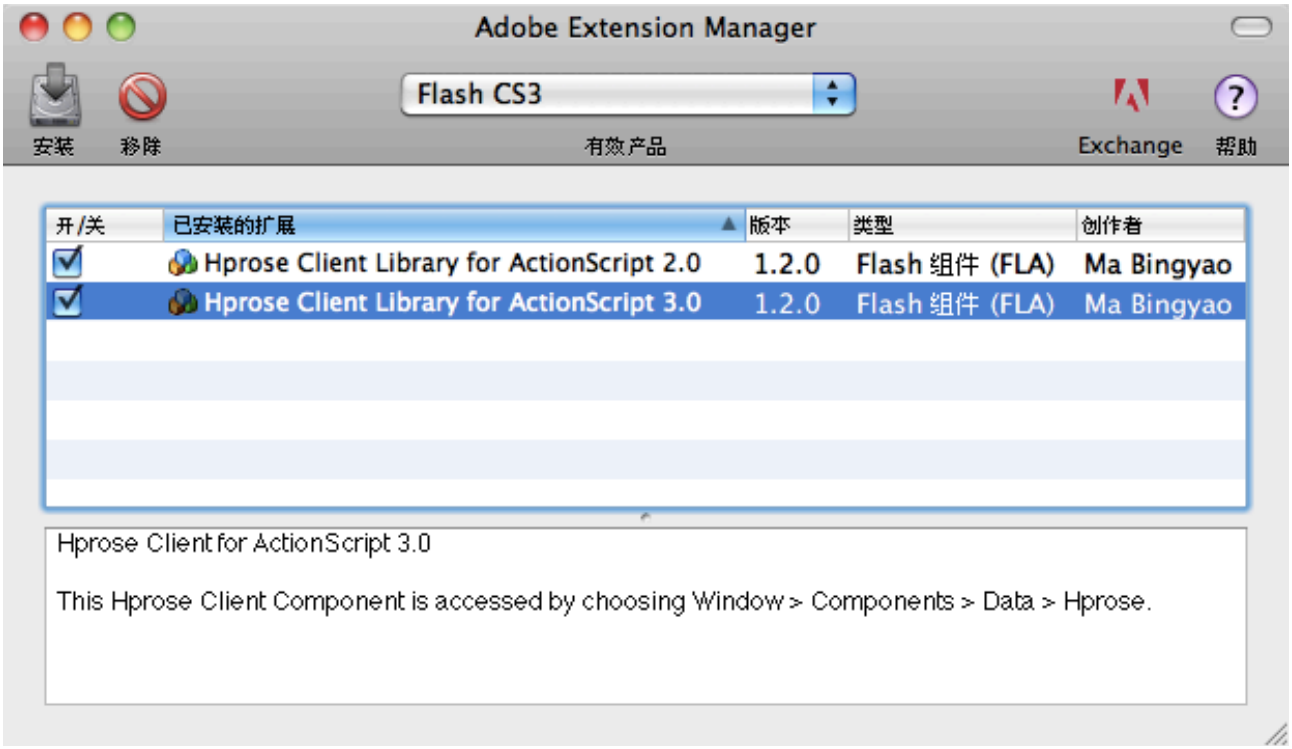
安装方法

为 Flash 安装 Hprose

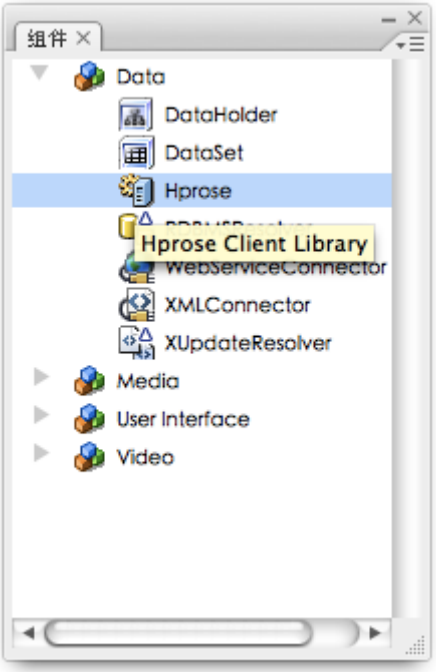
这两个版本都提供了 mxp 组件安装包，如果您已经安装了 Adobe Extension Manager 的话，直接双击 mxp 安装包即可完成安装。

需要注意一点，如果您使用的是 Adobe Flash CS4 之前的版本，需要安装的是 hprose_as2.mxp 和 hprose_as3.mxp 这两个安装包，如果您使用的是 Adobe Flash CS4 或者更高版本，则需要安装 hprose_as2_cs4.mxp 和 hprose_as3_cs4.mxp 这两个安装包。如果选择错误，安装后在组件面板上会找不到 Hprose 客户端组件。

ActionScript 2.0 和 ActionScript 3.0 版本的 Hprose 可以同时安装，不会发生冲突。安装之后，您将会在 Adobe Extension Manager 中看到这两个组件。



当您在 Flash 中新建一个应用后，在组件面板的 Data 分组中您就可以找到 Hprose 的客户端组件：

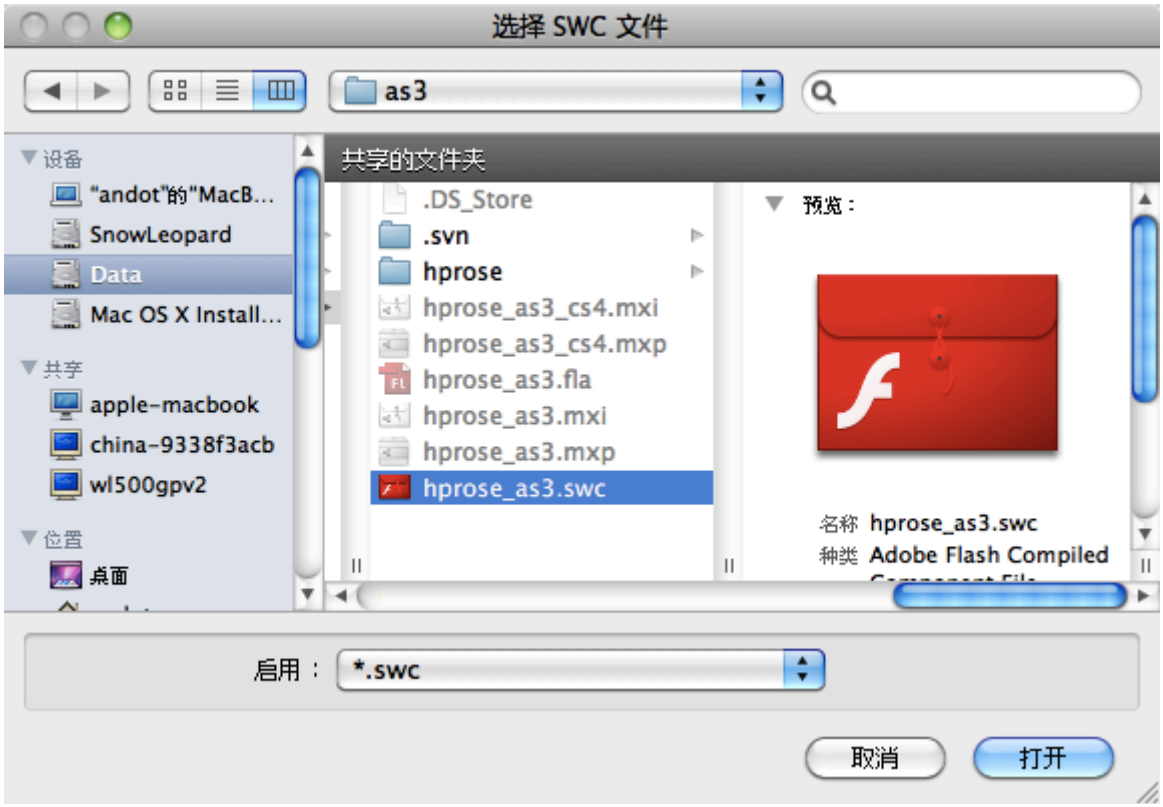


双击之后，就可以添加到您的程序中了。

为 Flex 安装 Hprose

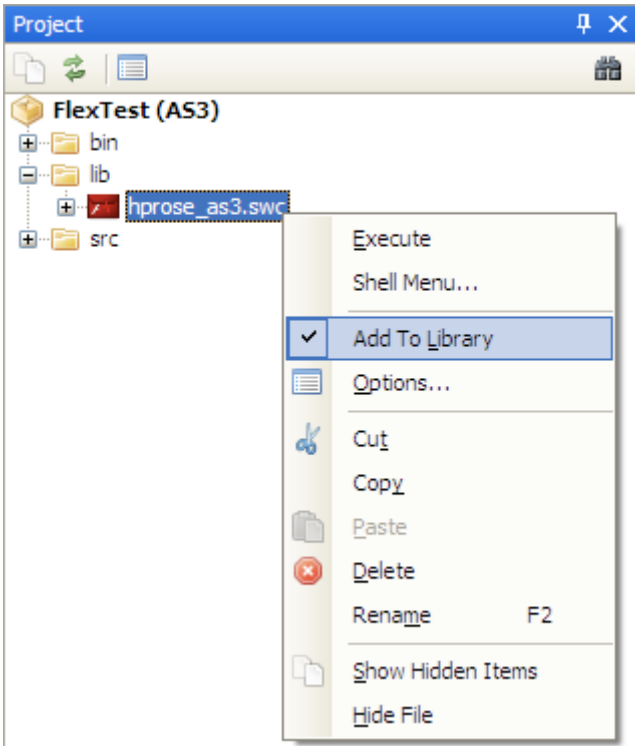
除了提供了 mxp 安装包之外，Hprose for ActionScript 还直接提供了 swc 的组件包。

在 Flex/Flash Builder 中 ,新建 Flex 项目 ,在选择库时点击添加 swc 按钮 ,选择 hprose_as3.swc 即可：

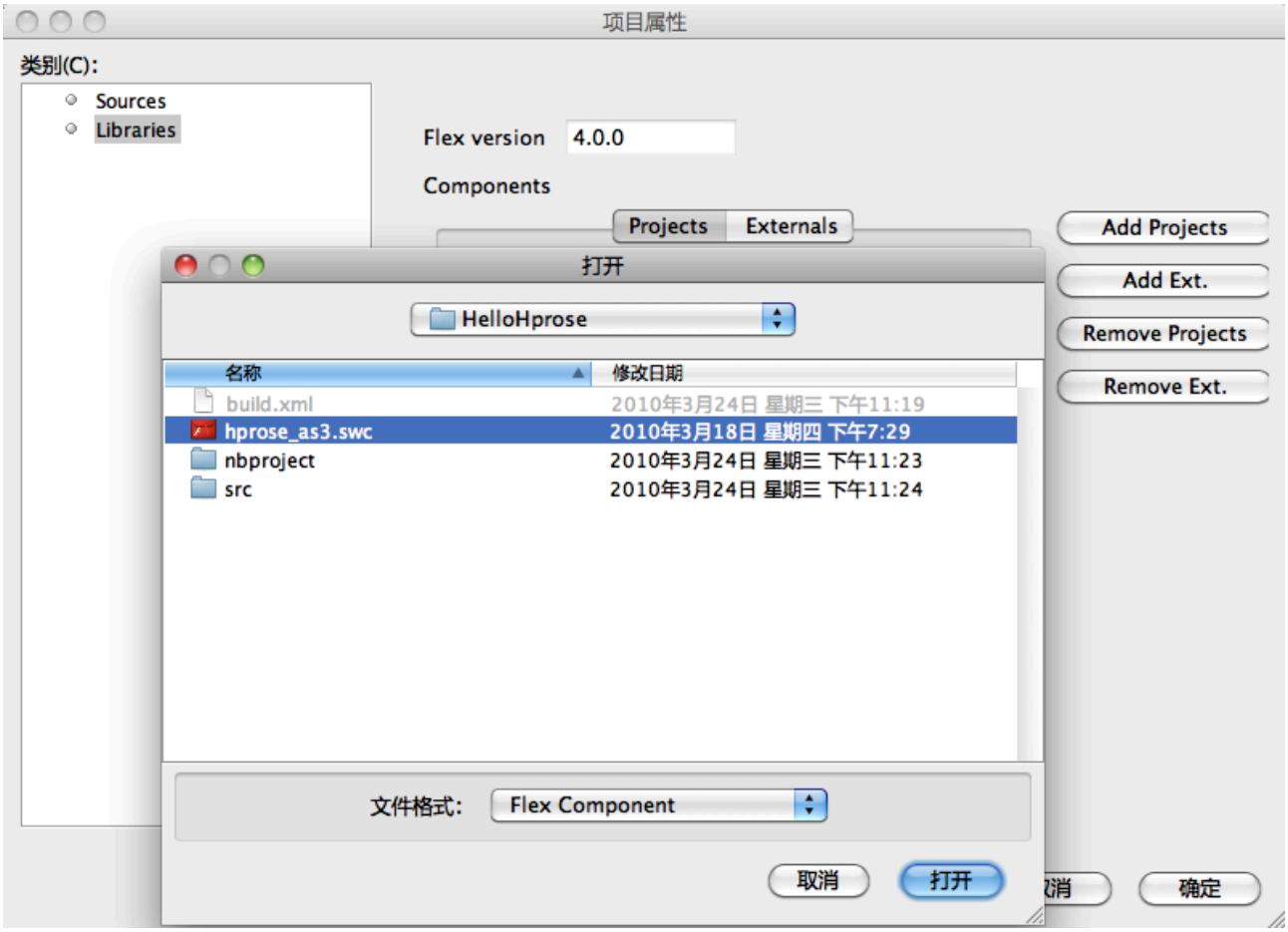


在 FlashDevelop 中创建 Flex 项目后，直接复制 hprose_as3.swc 到 Flex 项目的 lib 目录下，然后在项

目中，选择 hprose_as3.swc 之后点鼠标右键，选择 Add To Library。如下图所示：



在 Netbeans 中使用 FlexBean 插件创建 Flex Application 应用后，将 hprose_as3.swc 复制到 Flex 项目目录下。在项目上点鼠标右键，选择属性，之后点击添加扩展按钮选择 hprose_as3.swc 即可。如下图所示：



为 HaXe 安装 Hprose

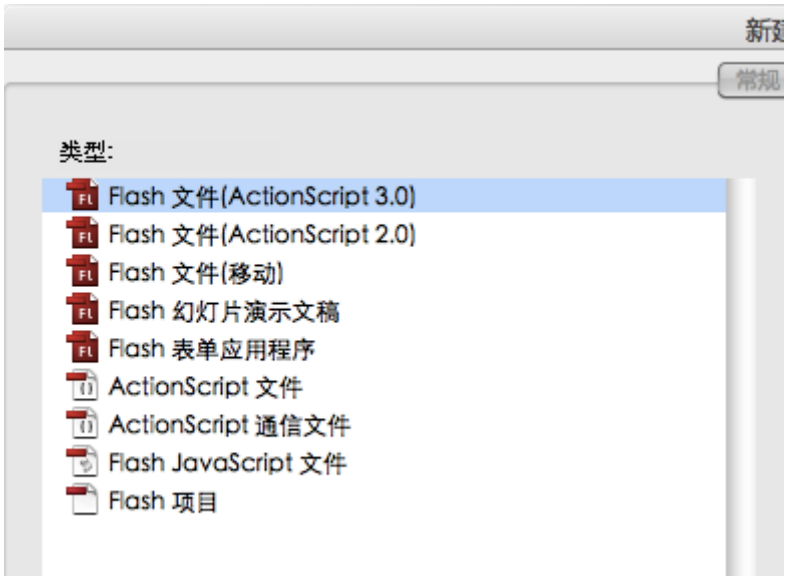
1. 将 hprose_as3.swc 改名为 hprose.zip。
2. 解压缩 hprose.zip，并找到解压之后的 library.swf。
3. 将 library.swf 改名为 hproselib.swf。
4. 执行 haxe --gen-hx-classes hproselib.swf。
5. 将生成的 hxclasses 中的 hprose 目录复制到您的 HaXe 工程的 src 中。
6. 将 hproselib.swf 复制到您的工程中。
7. 将 hproselib.swf 设置为库。
8. 接下来就可以使用 hprose 开发 HaXe 应用程序啦。

在 Flash 中创建 Hprose 的 Hello 客户端

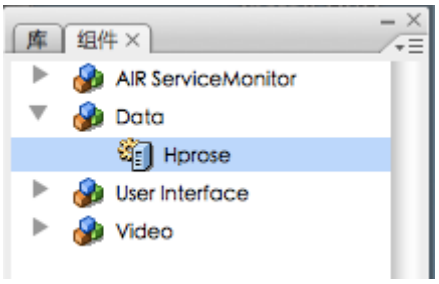
Hprose for ActionScript 的 1.2 版本只提供了客户端功能，因此我们还需要一个服务器才能进行讲解和演示。不过这并不是什么问题，Hprose 的客户端和服务器所使用的语言是没有必然联系的，不论服务器是 Java、C#还是 PHP，对客户端来说都是一样的。所以这里我们将使用 PHP 版本的 Hprose 服务器为例来进行讲解。为了方便大家可以直接使用本书中的例子进行练习，我们直接将服务器放在了官方网站上，地址为：http://www.hprose.com/example/，这是一个实际可用的服务，后面所有的例子，都以这个地址所提供的服务为例来进行说明。

另外，因为 Hprose for ActionScript 2.0 和 Hprose for ActionScript 3.0 在用法上没有任何区别，仅在功能支持上略有不同，在第一章中我们已经介绍了它们的一些不同之处，在后面的章节我们还会再提及一些它们的差别。为了节省篇幅，我们后面全部使用 Hprose for ActionScript 3.0 为例来进行讲解。

在 Flash 中使用 Hprose 非常简单，首先确认已经安装好了 Hprose，然后选择“文件→新建”，选择 Flash 文件（ActionScript 3.0）。



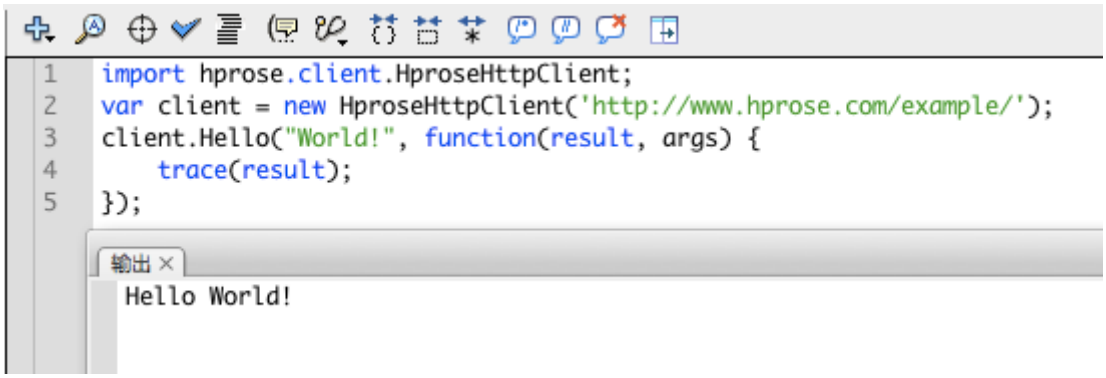
确定之后，打开组件面板，在“Data→Hprose”上双击，然后 Hprose 组件就被添加到您的程序库中了：



接下来打开动作面板，输入以下代码：

```
import hprose.client.HproseHttpClient;
var client = new HproseHttpClient('http://www.hprose.com/example/');
client.Hello("World!", function(result, args) {
    trace(result);
});
```

然后选择“控制→测试影片”，之后如果您在输出窗口中看到：



就说明已经成功了。

接下来我们来看看在 Flex 中怎么用。

在 Flex 中创建 Hprose 的 Hello 客户端

Flex 的开发工具有很多，比如 Flex/Flash Builder，Netbeans + Flexbean，FlashDeveop 等。但是不管在什么开发工具里，使用 Hprose 开发 Flex 应该的方法是一样的。

首先新建一个 Flex 项目，之后设置好 Hprose 库的路径（参见第一章安装部分）。然后将默认的 mxml 改成如下内容：

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:mx="library://ns.adobe.com/flex/mx"
               xmlns="hprose.client.*"
               minWidth="600" minHeight="480">
    <fx:Declarations>
        <HproseHttpClient id="client" uri="http://www.hprose.com/example/" />
    </fx:Declarations>
```

```
    <fx:Script>
        <![CDATA[
            import mx.controls.Alert;
            private function helloWorld():void {
                client.Hello("World!", function(result:*, args:Array):void {
                    Alert.show(result);
                });
            }
        ]]>
    </fx:Script>
    <s:Button label="Hello" click="helloWorld()" />
</s:Application>
```

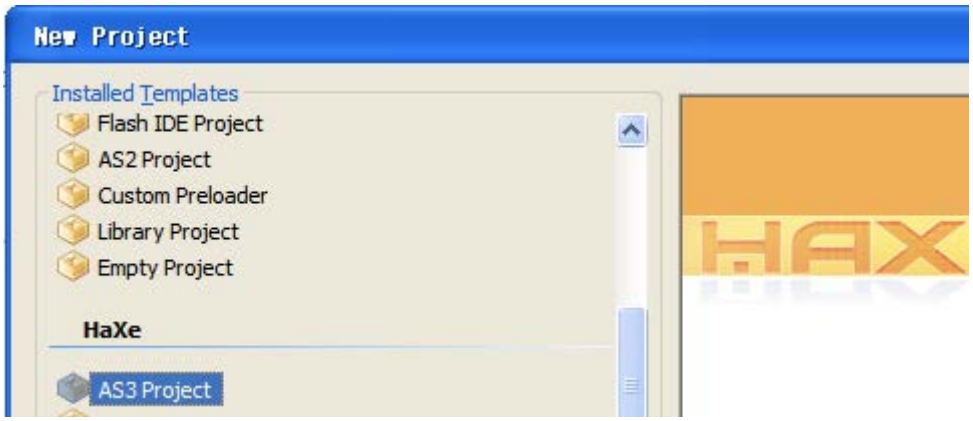
然后，保存并运行。之后在浏览器（或独立的 Flash 播放器）中点击 Hello 按钮：

如果您看到上面的画面 ,就说明客户端创建成功了 !上面是 Flex 4.0 的写法 ,如果您使用的是 Flex 3.0，只需稍作修改即可，这里不再单独举例。

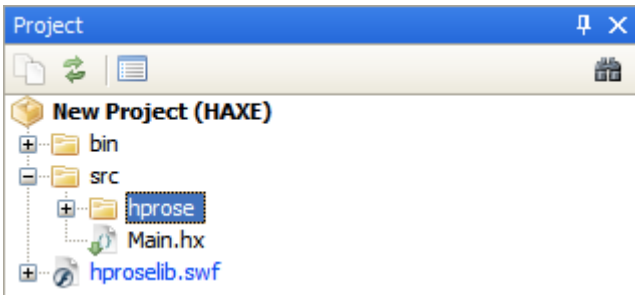
在 HaXe 中创建 Hprose 的 Hello 客户端

HaXe 是一个可以创建不同语言项目的语言，在它当中您可以直接使用 Hprose for ActionScript 来进行程序开发。下面我们来看一下如何使用 HaXe 和 Hprose 创建的 HelloWorld 实例。

这里我们以 FlashDevelop 作为 HaXe 开发环境。首先选择“Project→New Project…”, 然后选择 HaXe 的 AS3 Project：



接下来将 hproselib.swf 复制到项目目录下，将 hprose 的 hx 头文件复制到 src 目录下。



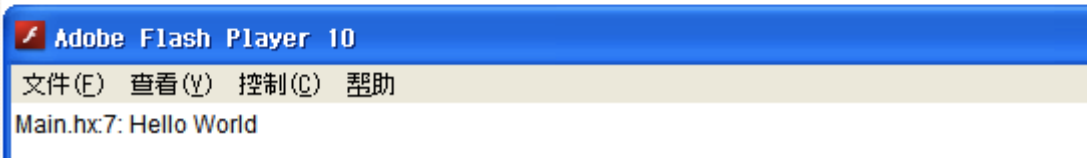
接下来就是写代码了：

```
import hprose.client.HproseHttpClient;
class Main {
    static public function main() {
        var client = new HproseHttpClient("http://www.hprose.com/example/");
        client.invoke("Hello", "World", function(result, args) {
            trace(result);
        });
    }
}
```

接下来按 F5 键，就可以看到运行结果了：

```
import hprose.client.HproseHttpClient;

class Main {
    static public function main() {
        var client = new HproseHttpClient("http://www.hprose.com/example/");
        client.invoke("Hello", "World", function(result, args) {
            trace(result);
        });
    }
}
```



上面的程序很简单，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 ActionScript 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 ActionScript 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

ActionScript 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 ActionScript 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

ActionScript 类型	Hprose 类型
Number、uint 类型中的整数（-2147483648 ~ 2147483647）、int	整型
纯数字字符串、uint 中整数范围之外的数字	长整型
Number 类型中的浮点数	浮点型
NaN	非数
Infinity	正无穷大
-Infinity	负无穷大
true	布尔真
false	布尔假
undefined，null	空
单字符字符串	字符
ByteArray	二进制型
字符串	字符串型
Date 类型	日期/时间/日期时间型

反序列化类型映射

Hprose 类型	ActionScript 类型
整型	int（ActionScript 3.0），Number 类型（ActionScript 2.0）
长整型	纯数字 String 类型
浮点型	Number 类型

Hprose 类型	ActionScript 类型
非数	Number 类型的 NaN
正无穷大	Number 类型的 Infinity
负无穷大	Number 类型的-Infinity
布尔真	true
布尔假	false
空	null
空串	""
二进制型	ByteArray (ActionScript 3.0), 不支持 (ActionScript 2.0)
字符/字符串型	String 类型
日期/时间/日期时间型	Date 类型

注意：ActionScript 2.0 不支持二进制型数据，但 ActionScript 3.0 支持！

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

ActionScript 中的 Array 类型数据被映射为 Hprose 列表类型。例如：

```
var a1:Array = new Array(1, 2, 3, 4, 5);
var a2:Array = new Array(3);
var a3:Array = ['Tom', new Date(1982, 2, 23), 28, 'Male'];
```

这些数组都被映射为 Hprose 列表类型 ,数组元素允许是任意可以序列化的类型。但是要避免这样使用：

```
var a:Array = [];
a[10000000] = 'foo bar';
```

因为这在传输时 ,被序列化后的数据将有 10MB 之多 ,而这其中却只有几个字节的数据是您所关心的。那如果需要使用这种非连续下标的数组该怎么处理呢？

很简单，使用下面这种方式就可以啦。

```
var o:Object = {};
o[10000000] = 'foo bar';
```

这里的 o 与上面的 a 的不同之处在于，o 不是一个数组，而是一个对象，但是您仍然可以以索引方式来存取它的元素。

那为何使用这种方式就可以避免大量冗余数据传输了呢？

因为对象是采用下面这种字典类型方式序列化的。

字典类型

ActionScript 中的 Object 类型数据被映射为 Hprose 字典类型。例如：

```
var user1:Object = {'name': 'Tom',
                    'birthday': new Date(1982, 2, 23),
                    'age': 28,
                    'sex': 1,
                    'married': true };
```

注意，不要把 Array 类型的对象当 Object 类型使用，例如：

```
var user2:Array = [];
user2['name'] = 'Tom';
user2['birthday'] = new Date(1982, 2, 23);
user2['age'] = 28;
user2['sex'] = 1;
user2['married'] = true;
```

这个数据是无法按照您期望的方式传输给服务器的，它的所有属性在传输时都会被完全忽略，只会传递一个空数组到服务器端，这是因为数组始终是按照它的 length 属性所表示的长度来传递的。

另外，在 ActionScript 3.0 中，flash.utils 下的 Dictionary 类型也是按照字典类型序列化的。

在反序列化时，ActionScript 3.0 都是按照 flash.utils 下的 Dictionary 类型来对字典类型反序列化的，而 ActionScript 2.0 都是按照 Object 类型进行反序列化。

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

通过定义类来定义可序列化类型

ActionScript 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 ActionScript 3.0 中可以这样创建自定义类型：

```
package {
```

```
    public class User {
        private var _name:String;
        private var _birthday:Date;
        private var _sex:int;
        private var _age:int;
        private var _married:Boolean;
        public function User() {
        }
        public function get name():String {
            return _name;
        }
        public function set name(value:String):void {
            _name = value;
        }
        public function get birthday():Date {
            return _birthday;
        }
        public function set birthday(value:Date):void {
            _birthday = value;
        }
        public function get sex():int {
            return _sex;
        }
        public function set sex(value:int):void {
            _sex = value;
        }
        public function get age():int {
            return _age;
        }
        public function set age(value:int):void {
            _age = value;
        }
        public function get married():Boolean {
            return _married;
        }
        public function set married(value:Boolean):void {
            _married = value;
        }
    }
}
```

上面这段代码就定义了一个 User 类，下面我们可以这样用它创建一个 User 对象：

```
var user3:User = new User();
user3.name = 'Tom';
```

```
user3.birthday = new Date(1982, 2, 23);
user3.age = 28;
user3.sex = 1;
user3.married = true;
```

这样这个对象，就会按照自定义 User 类型来进行序列化传输了。

如果您的类中的属性全部都是像上面那样直接对字段进行存取，那么可以化简为：

```
package {
    public class User {
        public var name:String;
        public var birthday:Date;
        public var sex:int;
        public var age:int;
        public var married:Boolean;
        public function User() {
        }
   }
}
```

自定义类中的公开的属性/字段名，映射为 Hprose 对象类型中的属性名，自定义类中的公开的属性/字段值，映射为 Hprose 对象类型中的属性值。对于所有私有和保护的属性/字段，以及类型为方法类型的属性/字段，序列化时会全部忽略。

如果您希望字段全部是动态的，那么还可以进一步化简为：

```
package {
    public dynamic class User {
        public function User() {
        }
    }
}
```

在定义上面这些类时需要注意，类构造函数必须是无参的，或者所有参数都包含默认参数。

如果使用类定义序列化类型，那么可以使用包名。包名在序列化时作为类型名的一部分。在序列化后的类型名中，包名之中的点分隔符会被替换为下划线分隔符，包名和类名之间也用下划线分隔符连接。

通过定义函数来定义可序列化类型

如果只是需要接收一个从服务器端返回的对象，我们可以不需要在客户端定义这个对象类型，Hprose for ActionScript 会自动在接受数据时，创建这个类型。但是它所自动创建的类型与上面我们的几种定义方式的并不相同，也就是说，除了可以通过类来定义可序列化类型以外，Hprose for ActionScript 还提供了另一种自定义可序列化类型的方式——函数方式，例如：

```
function User():void {
    this.getClassName = function():String { return 'User'; };
```

```
}
```

但这样的话，在声明 User 的对象变量时，要像下面这样使用 Object 类型声明，因为 User 在这里不是一个类，而是一个函数。

```
var user4:Object = new User();
user4.name = 'Tom';
user4.birthday = new Date(1982, 2, 23);
user4.age = 28;
user4.sex = 1;
user4.married = true;
```

但是在序列化传输时，它们都是按照相同的格式传输的。

使用函数来定义序列化类型时，函数中必须要有一个 getClassName 方法，其返回值是类型名，也就是说，它如果是由好几部分构成的话，必须要用下划线分隔符分割，而不能用点分隔符分割。使用函数方式定义序列化类型时，序列化的类型名与函数名无关，只与 getClassName 方法的返回值有关。

Hprose for ActionScript 在对返回值中的自定义类型对象的未知类进行自动定义就是采用的这种方式。

直接定义可序列化对象

有时候我们并不想定义一个函数来创建自定义可序列化类型，Hprose 允许您直接创建一个可序列化类型的对象而不需要定义类或函数：

```
var user5:Object = {'getClassName': function():String { return 'User'; },
                    'name': 'Tom',
                    'birthday': new Date(1982, 2, 23),
                    'age': 28,
                    'sex': 1,
                    'married': true};
```

您发现了，这个跟前面介绍字典类型时的例子很相似，但是它多了一个方法属性：getClassName，这个函数的返回值就是这个自定义类型的类型名。这样，它就不再作为一个字典类型进行序列化，而是作为一个对象类型进行序列化了。

通过 ClassManager 来注册自定义类型

通过 class 方式定义的类，要跟其它语言交互时，需要包名和类名都要跟其他语言的定义匹配，通过 function 方式定义的类型，又需要有一个 getClassName 方法。有没有办法让已有的类或 function 类型在不需要任何修改的情况下，就能跟其它语言中的类型交互呢？

通过 hprose.io.ClassManager 的 register 方法就可以轻松实现这个需求。

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

```
ClassManager.register(my.package.User, 'User');
```

假如您有一个用 function 定义的 User 类型，也可以通过这种方式注册：

```
ClassManager.register(User, 'User');
```

这种情况下，您不需要为 User 添加 getClassName 方法。

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 ActionScript 客户端，在本章中您将深入的了解 Hprose for ActionScript 客户端的更多细节。

本章提要

- 基本远程调用
- 高级远程调用
- 异常处理
- 超时设置
- HTTP 参数设置
- 跨域调用

基本远程调用

可变的参数和结果类型

本章的例子我们都以 Flex 为例来讲解。

前面在快速入门中，我们的 Flex 版的 HelloWorld 实例里面的 Hprose 客户端对象是通过 mxml 标签创建的，并且整个实例是按照 Flex 4.0 的标签语法写的，下面的例子中我们直接使用脚本来创建，并且使用 Flex 3.0 的标签：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
      import hprose.client.HproseHttpClient;
      private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/ex
ample/');
      private function sum1():void {
         client.sum(1, 2, 3, 4, 5, function(result:*, args:Array):void {
            trace(result);
            trace(args);
         });
      }
      private function sum2():void {
         client.sum(1.2, 3.4, 5.6, function(result:*, args:Array):void {
            trace(result);
            trace(args);
         });
      }
      private function sum3():void {
         client.sum('123456', '789', function(result:*, args:Array):void {
            trace(result);
            trace(args);
         });
      }
      ]]>
   </mx:Script>
   <mx:Button label="sum1" click="sum1()" />
   <mx:Button label="sum2" click="sum2()" />
   <mx:Button label="sum3" click="sum3()" />
</mx:Application>
```

这个例子中，我们直接在脚本中创建了 HproseHttpClient 的对象 client。但是需要注意的是，在 Flex 中，只能在函数中使用 client 对象，而不能直接在全局直接调用 client 上的任何方法，否则编译时会发生错

误，不能通过。

因为本例中有多个输出，如果使用 Alert 的话，会比较乱，所以这里我们使用 trace 进行输出。程序运行后，分别点击 sum1、sum2、sum3 按钮，会看到如下结果：

因为 Hprose 是支持动态参数类型传递的，所以这个例子中我们对远程方法 sum 分别带入了不同个数，不同类型的参数，但是都得到了正确的结果。这是其它静态类型绑定的远程调用技术所做不到的。

异步顺序调用

下面我们再来看一下如果这三次调用放在一起会是什么结果：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import hprose.client.HproseHttpClient;
        private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/example/');
        private function sum():void {
            client.sum(1, 2, 3, 4, 5, function(result:*, args:Array):void {
                trace(result);
                trace(args);
            });
            client.sum(1.2, 3.4, 5.6, function(result:*, args:Array):void {
                trace(result);
                trace(args);
            });
            client.sum('123456', '789', function(result:*, args:Array):void {
                trace(result);
                trace(args);
            });
        }
        ]]>
    </mx:Script>
    <mx:Button label="sum" click="sum()" />
</mx:Application>
```

程序运行后，点击 sum 按钮，您会发现结果可能是这样的：

```
124245
123456,789
15
1,2,3,4,5
10.2
1.2,3.4,5.6
```

也就是说，结果的返回顺序并没有按照我们书写的顺序执行。这是因为在 Flex 中，所有的 Hprose 远程调用都是异步的。那么如何让异步调用能够保证按顺序返回结果呢？我们可以用下面的方式：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
      import hprose.client.HproseHttpClient;
      private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/ex
ample/');
      private function sum():void {
         client.sum(1, 2, 3, 4, 5, function(result:*, args:Array):void {
            trace(result);
            trace(args);
            client.sum(1.2, 3.4, 5.6, function(result:*, args:Array):void {
               trace(result);
               trace(args);
               client.sum('123456', '789', function(result:*, args:Array):void {
                  trace(result);
                  trace(args);
               });
            });
         });
      }
      ]]>
   </mx:Script>
   <mx:Button label="sum" click="sum()" />
</mx:Application>
```

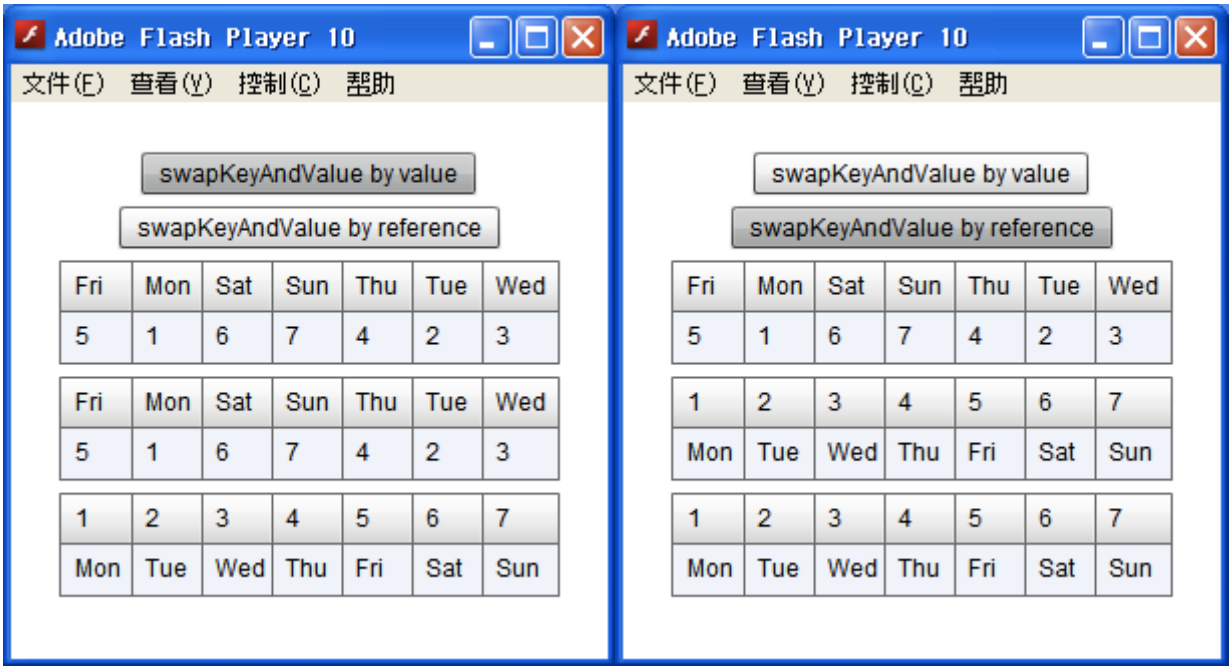
引用参数传递

下面这个例子很好的说明了如何进行引用参数传递：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
```

```
        <![CDATA[
        import hprose.client.HproseHttpClient;
        private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/ex
ample/');
        private function swapKeyAndValue(byref:Boolean):void {
            var arg:Object = {"Mon": 1, "Tue": 2, "Wed": 3, "Thu": 4, "Fri": 5, "Sat": 6, "Sun
": 7};
            client.swapKeyAndValue(arg, function(result:*, args:Array):void {
                arg1_dg.dataProvider = arg;
                arg2_dg.dataProvider = args[0];
                result_dg.dataProvider = result;
            }, byref);
        }
        ]]>
    </mx:Script>
    <mx:Button label="swapKeyAndValue by value" click="swapKeyAndValue(false)"  />
    <mx:Button label="swapKeyAndValue by reference" click="swapKeyAndValue(true)"  />
    <mx:DataGrid id="arg1_dg" rowCount="1" width="100%" />
    <mx:DataGrid id="arg2_dg" rowCount="1" width="100%" />
    <mx:DataGrid id="result_dg" rowCount="1" width="100%" />
</mx:Application>
```

运行界面：



上面例子中，arg 是调用时的参数，回调函数的第二个参数 args 是调用后的参数列表，回调函数之后，还有一个参数是表示是否进行引用参数传递，如果为 true 就表示引用参数传递，否则为非引用参数传递，默认为非引用参数传递。

Hprose 1.2 for ActionScript 中，客户端对象上增加了一个 byref 属性，您可以通过它来设置是否为引

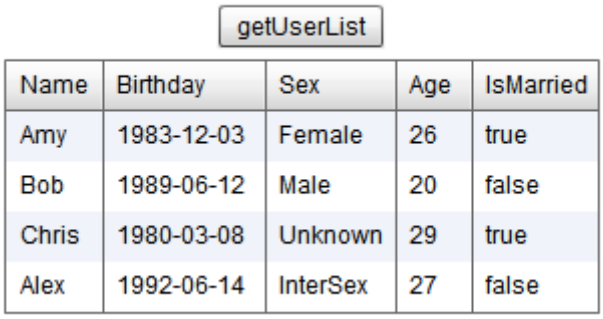
用参数传递的默认值。但通常您不需要改变它，除非您所有的调用都是引用参数传递。另外，引用参数传递会消耗更多带宽，因此，如果没有必要，最好不要采用引用参数传递。

自定义类型传递

下面我们来看一个传递自定义对象列表的例子：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
      import hprose.client.HproseHttpClient;
      private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/example/');
      private function getUserList():void {
         client.getUserList(function(result:Array, args:Array):void {
            dg.dataProvider = result;
         });
      }
      private function dateFormat(item:Object, column:DataGridColumn):String {
         return df.format(item.birthday);
      }
      private function sexFormat(item:Object, column:DataGridColumn):String {
         return ['Unknown', 'Male', 'Female', 'InterSex'][item.sex];
      }
      ]]>
   </mx:Script>
   <mx:Button label="getUserList" click="getUserList()"/>
   <mx:DateFormatter id="df" formatString="YYYY-MM-DD"/>
   <mx:DataGrid id="dg">
      <mx:columns>
         <mx:DataGridColumn dataField="name" headerText="Name"/>
         <mx:DataGridColumn dataField="birthday" headerText="Birthday" labelFunction="dateFormat"/>
         <mx:DataGridColumn dataField="sex" headerText="Sex" labelFunction="sexFormat"/>
         <mx:DataGridColumn dataField="age" headerText="Age"/>
         <mx:DataGridColumn dataField="married" headerText="IsMarried"/>
      </mx:columns>
   </mx:DataGrid>
</mx:Application>
```

该程序运行界面如下：



该程序从服务器端返回一个自定义 User 类型对象的列表，并将结果显示在 DataGrid 中，在这里，我们并没有定义 User 类，所以 Hprose 会自动帮我们生成。

通过上面两个例子，我们会发现 Hprose 的返回结果可以直接绑定到 Flex 的控件上，非常方便。

高级远程调用

通过 invoke 方法进行远程调用

除了可以向上面那样直接通过方法名进行远程调用外，Hprose for ActionScript 还提供了通过 invoke 方法进行远程调用。在快速入门一章中，在 HaXe 中创建 Hprose 客户端的例子就是使用 invoke 方法来进行远程调用的，因为在 HaXe 中只支持使用 invoke 方法来进行远程调用。在 Flash 和 Flex 中，我们也可以使用 invoke 来进行远程调用，这通常在方法未知，或者动态调用时用到。您只需要将调用的远程方法名，作为 invoke 方法的第一个参数，远程调用的参数从第二个参数开始就可以啦。这里不再举例。

远程调用事件

Hprose for ActionScript 除了可以让您直接在调用时指定回调函数之外，您还可以通过事件的方式来指定回调动作，对于每次调用您可以指定 3 个事件，他们分别是：

1. hprose.client.HproseSuccessEvent
2. hprose.client.HproseErrorEvent
3. flash.events.ProgressEvent（在 ActionScript 2.0 版本中是 hprose.client.HproseProgressEvent）

下面我们来看一下用法：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
      import flash.events.ProgressEvent;
      import hprose.client.HproseErrorEvent;
      import hprose.client.HproseHttpClient;
      import hprose.client.HproseHttpInvoker;
      import hprose.client.HproseSuccessEvent;
      private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/ex
```

```
ample/');
       private function Hello():void {
          var hello:HproseHttpInvoker = client.hello("Hprose");
          hello.addEventListener(HproseSuccessEvent.SUCCESS, function(event:HproseSuccessE
vent):void {
             trace(event);
          });
          hello.addEventListener(HproseErrorEvent.ERROR, function(event:HproseErrorEvent):
void {
             trace(event);
          });
          hello.addEventListener(ProgressEvent.PROGRESS, function(event:ProgressEvent):voi
d {
             trace(event);
          });
          hello.start();
       }
       ]]>
   </mx:Script>
   <mx:Button label="Hello" click="Hello()"/>
</mx:Application>
```

这个例子的输出是：

```
[ProgressEvent type="progress" bubbles=false cancelable=false eventPhase=2 bytesLoaded=19
bytesTotal=0]
[HproseSuccessEvent type="success" bubbles=false cancelable=false eventPhase=2 result="Hel
lo Hprose" args=Hprose]
```

这里我们在调用 client 上的 Hello 后，并没有指定回调函数。这时，这个远程方法 Hello 并不是立即开始执行，而是返回一个 HproseHttpInvoker 对象，这时，我们可以给这个对象添加事件。HproseSuccessEvent 事件在调用成功时被执行，HproseErrorEvent 事件在调用出错时被执行，这两个事件不会同时发生。ProgressEvent（或 HproseProgressEvent）事件在调用过程中被执行，它通常用来显示结果返回的进度。需要注意的是：如果是要上传大量数据到服务器，这个事件是无法监控上传进度的。

您可以用 addEventListener 对同一个事件添加多个回调函数，也还可以用 removeEventListener 来移除某个事件的某个回调函数。

最后，只有执行 HproseHttpInvoker 对象上的 start 方法，调用才会真正开始执行。

错误处理和进度处理的回调方式

在 Hprose 1.2 for ActionScript 及其之后的版本中，错误处理和进度处理也可以通过回调方式作为参数直接传入远程调用中。

错误处理的回调函数参数有两个，第一个表示远程方法名，第二个表示异常对象。

进度处理的回调函数参数也是两个，第一个表示 bytesLoaded，第二个表示 bytesTotal。

回调参数的顺序是：成功回调，出错回调和进度回调。出错回调和进度回调皆可以忽略。也可以只写成功回调和出错回调。

HproseHttpInvoker

HproseHttpInvoker 除了可以指定和移除事件，还提供了终止调用，设置引用参数传递，获取结果、获取返回参数等功能。

如果在远程方法执行过程中希望终止调用，可以调用 HproseHttpInvoker 对象上的 stop 方法。

在使用 HproseHttpInvoker 事件方式调用时，要设置引用参数传递，只需要设置 HproseHttpInvoker 对象的 byRef 属性为 true 即可，这个属性需要在调用 start 方法之前设置。

有时候您需要在回调函数之外使用返回结果，那么您可以通过 getResult 方法来获取，如果您需要获取引用参数传递后的参数，可以通过 getArguments 方法获取。

当您不确定调用是否完成时，可以通过 isCompleted 方法来判断。当您不确定调用在完成后是否成功时，可以通过 isSuccess 方法来判断。

远程代理对象

除了您可以在 HproseHttpClient 对象上进行远程调用之外，您还可以在 HproseHttpProxy 的远程代理对象上进行远程调用。实际上 HproseHttpClient 只是一个特别的 HproseHttpProxy 对象。

那究竟 HproseHttpProxy 是什么呢？它有什么作用？又是怎样使用呢？下面我们就来为您解开这些疑惑。

在 Hprose 中，服务器端可以直接发布对象，这样该对象上所有的公开方法都会被发布，也可以直接发布类，这样该类上所有的静态公开方法也都会被发布，但如果我们同时发布多个对象上的方法时，方法名可能会存在冲突，这时可以通过别名机制，让每一个类上的方法都有一个别名前缀，这个别名前缀通常是对象名或者类名。这样在客户端就可以通过带有别名前缀的全名进行远程调用了。但是直接在客户端使用带有别名前缀的远程方法并不是很方便，因为每次都需要写别名前缀，所以就有了 HproseHttpProxy，它可以让某个别名前缀变成一个远程代理对象，之后您可以在这个代理对象上面使用正常的方法名进行调用，而不再需要别名前缀。

这样说您可能还是不明白 ,没关系我们来举一个实际一点的例子。例如服务器端我们发布了一个 exam1 对象和一个 exam2 对象，这两个对象都是同一个 Exam 类的不同实例，所以它们拥有相同的方法名，因此我们发布时需要为它们分别指定 exam1 和 exam2 这两个别名前缀，假设 Exam 类上定义了 login、add、update、delete 这四个公开方法。那么我们在客户端需要调用 exam1 的这四个方法时，使用 HproseHttpClient 是应该这样来调用的：

```
client.exam1_login('admin', '123456', function(result:*, args:Array) {
    trace(result);
}
client.exam1_add(user, function(result:*, args:Array) {
    trace(result);
}
client.exam1_update(12, user2, function(result:*, args:Array) {
    trace(result);
}
client.exam1_delete(8, function(result:*, args:Array) {
```

```
        trace(result);
    }
```

但是您会发现 ,这样每次都需要写 exam1_这个前缀 ,比较麻烦 ,那么我们来看看使用 HproseHttpProxy 之后怎样写：

```
    var exam1:HproseHttpProxy = client.exam1;
    exam1.login('admin', '123456', function(result:*, args:Array) {
        trace(result);
    }
    exam1.add(user, function(result:*, args:Array) {
        trace(result);
    }
    exam1.update(12, user2, function(result:*, args:Array) {
        trace(result);
    }
    exam1.delete(8, function(result:*, args:Array) {
        trace(result);
    }
```

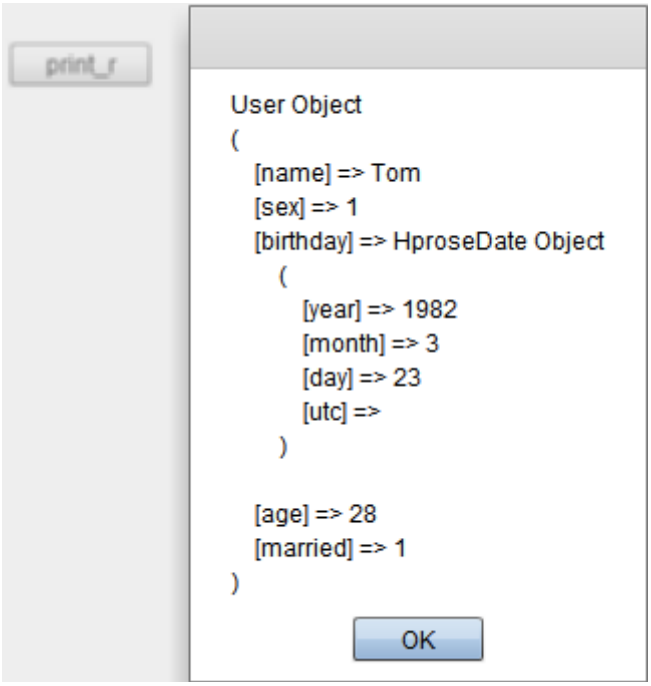
您会发现 ,client.exam1 直接作为一个 HproseHttpProxy 对象返回了 ,并且在后面 ,我们直接在 exam1 对象上调用 login、add、update 和 delete 这四个方法，而不再需要加别名前缀了。

不过我们的实例服务器上并没有提供一个 exam1 这样的类让您练习这种用法，但我们的服务器上却有发布 php 的 print_r 这个内置函数，这个函数虽然不是什么带有别名前缀的方法名，可是它却正好是下划线分割的，那么我们是不是可以将 print_r 变成 print.r 来进行调用呢？我们来试试下面的例子：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>
        <![CDATA[
        import hprose.client.HproseHttpClient;
        import hprose.client.HproseHttpProxy;
        import mx.controls.Alert;
        private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/ex
ample/');
        private function print_r():void {
            var print:HproseHttpProxy = client.print;
            var user:Object = {'getClassName': function():String { return 'User'; },
                               'name': 'Tom',
                               'birthday': new Date(1982, 2, 23),
                               'age': 28,
                               'sex': 1,
                               'married': true};
            print.r(user, true, function(result:String, args:Array):void {
                Alert.show(result);
```

```
        });
    }
    ]]>
</mx:Script>
<mx:Button label="print_r" click="print_r()"/>
</mx:Application>
```

让我们来看看运行结果吧：



非常棒！我们成功了！

这个例子很简单，就是将 user 对象在服务器端通过 print_r 转换为字符串（请注意，我们这里带入了两个参数，第二个参数 true 表示 print_r 转换的字符串不输出而是作为结果返回。如果您对 print_r 的详细用法感兴趣 ,请参考 PHP 用户手册)并返回。这里我们将 print_r 的前半部分变成了一个远程代理对象 print，后面我们调用了 print 对象上的 r 方法，这个调用在服务器端被正确的映射到了 print_r 函数上并返回了正确结果。

没错，这就是 HproseHttpProxy 代理对象的用法。也就是说，所有形如：client.xxx_yyy_zzz 这样的调用，都可以写做为：client.xxx.yyy.zzz，所有的下划线分隔符（下划线并不限制个数，多少个都可以），都可以变成点分隔符，并且可以从任何一个点分隔符处分开，作为一个独立对象使用。

掌握了这些高级用法，相信会让您的 hprose 编程变得更加灵活简单。

异常处理

下面我们再来看一下 Hprose for ActionScript 中的异常处理吧。

因为 Hprose for ActionScript 只提供异步调用，所以如果调用过程中发生异常，异常将不会被抛出。而是通过错误事件的方式来通知用户。

前面一节，我们在介绍通过事件来进行远程调用时，提过可以为每个调用都指定一个单独的错误处理事件。但是如果每个调用都要提供错误处理事件还是很麻烦的一件事。所以 Hprose for ActionScript 提供

了一个统一的错误处理事件。这个统一的错误处理事件，只需要在 HproseHttpClient 对象上指定即可，例如：

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">
   <mx:Script>
      <![CDATA[
      import hprose.client.HproseErrorEvent;
      import hprose.client.HproseHttpClient;
      import hprose.client.HproseHttpProxy;
      import mx.controls.Alert;
      private var client:HproseHttpClient = new HproseHttpClient('http://www.hprose.com/ex
ample/');
      private function callMissingFunction():void {
         client.addEventListener(HproseErrorEvent.ERROR, function(event:HproseErrorEvent)
:void {
            trace(event.name);
            trace(event.error);
         });
         client.Hi(function(result:String, args:Array):void {
            Alert.show(result);
         });
      }
      ]]>
   </mx:Script>
   <mx:Button label="callMissingFunction" click="callMissingFunction()" />
</mx:Application>
```

因为服务器端并没有发布 Hi 函数，所以上面的程序运行后，会有下面的结果：

```
Hi
HproseException: Can't find this function hi().
file: D:\phpox\hprose\www\example\hproseHttpServer.php
line: 156
trace: #0 D:\phpox\hprose\www\example\hproseHttpServer.php(384): HproseHttpServer->doInvok
e()
#1 D:\phpox\hprose\www\example\hproseHttpServer.php(401): HproseHttpServer->handle()
#2 D:\phpox\hprose\www\example\index.php(60): HproseHttpServer->start()
#3 {main}
```

在 Hprose 1.2 for ActionScript 及其之后的版本中，如果您使用的是 Flex，那么该错误事件也可以通过 mxml 标签属性直接设置。

超时设置

Hprose 1.2 for ActionScript 及其之后的版本中增加了超时设置。只需要设置客户端对象上的 timeout 属性即可，单位为毫秒。当调用超过 timeout 的时间后，调用将被中止，并触发错误事件。默认超时时间为 30 秒（30000 毫秒）。

HTTP 标头设置

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 setHeader 方法就可以啦，该方法的第一个参数为标头名，第二个参数为标头值，这两个参数都是字符串型。如果将第二个参数设置为 null、undefined、0、false 或者空字符串，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。

跨域调用

Hprose for ActionScript 可以进行跨域调用，只需要在服务器端对跨域权限进行一下简单的设置即可。也就是说，需要在 Hprose 服务器的 Web 根目录上放置一个 crossdomain.xml 文件，如果您希望来自所有地方的用户都可以对您的服务进行调用的话，您可以将它设置为如下内容：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE cross-domain-policy SYSTEM
    "http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd" >
<cross-domain-policy>
    <site-control permitted-cross-domain-policies="all" />
    <allow-access-from domain="*" />
    <allow-http-request-headers-from domain="*" headers="*"/>
</cross-domain-policy>
```

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 ActionScript 2.0 中，序列化的结果以字符串方式表示，在 ActionScript 3.0 中，序列化的结果以 ByteArray 方式表示。用户可以通过 HproseFormatter.unserialize 方法来将该结果反序列化为普通模式的结果。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 类中定义。

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。该模式主要用于方便调试。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。

在 ActionScript 中，通常我们不需要普通模式以外的其他模式返回结果。因此，这里不再举例说明。