用户手册

1.3

（ASP 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 ASP 进行分布式程序开发的全过程。

本章提要

- 安装 Hprose for ASP
- 创建 Hprose 的 Hello 服务器
- 创建 Hprose 的 Hello 客户端

安装 Hprose for ASP

Hprose for ASP 是使用 JScript 编写的，但它支持 JScript 和 VBScript 两种语言。

它支持所有可以运行 ASP 的 IIS 服务器。

安装方法

Hprose for ASP 包含源码版本和压缩版本这两个版本。其中 hproseCommon.js，hproseIO.js，hproseHttpServer.js 以及 hproseHttpClient.js 这四个文件是 Hprose for JavaScript 的源码版本，通常您不需要直接使用它们，而 compressed 目录中的 hprose.js、hproseServer.js 和 hproseClient.js 是压缩版本。但如果是在程序调试阶段，使用压缩版本不方便调试的话，可以按以下顺序将源码版本包含到您的 html 中：

```
<script runat="server" language="JScript" type="text/javascript" src="hproseCommon.js"></s
cript>
<script runat="server" language="JScript" type="text/javascript" src="hproseIO.js"></scrip
t>
<script runat="server" language="JScript" type="text/javascript" src="hproseHttpServer.js"
></script>
<script runat="server" language="JScript" type="text/javascript" src="hproseHttpClient.js"
></script>
```

创建 Hprose 的 Hello 服务器

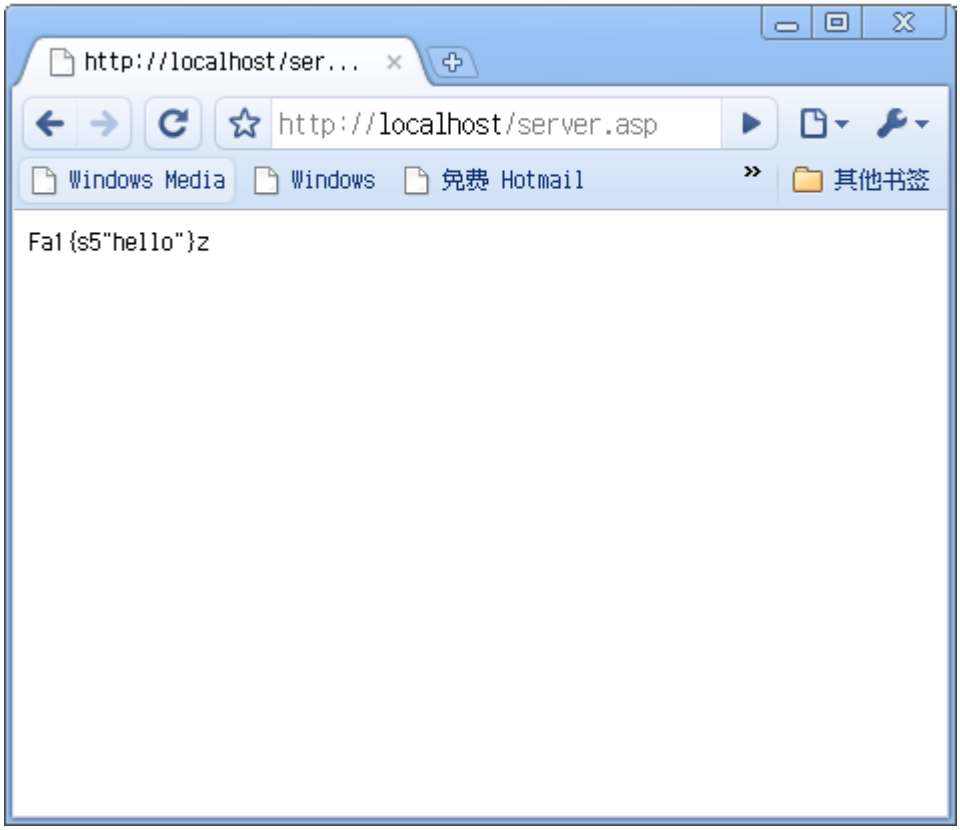
创建 ASP 的 Hprose 的服务器非常简单，下面我们以 IIS 环境为例来讲解。假设在 IIS 下已经配置好了 ASP，发布路径为 C:\Inetpub\wwwroot。然后我们在发布目录下建立一个 hprose 目录，把 Hprose for ASP 的压缩文件放到该目录下。然后在发布目录下创建一个 server.asp 文件，内容如下：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseServer.
js"></script>
<%
   Function hello(name)
      hello = "Hello " & name & "!"
   End Function

   Dim server
   Set server = HproseHttpServer.create()
   server.addFunction GetRef("hello")
   server.start
%>
```

这样，我们的服务器端就创建好了，是不是相当的简单啊？

好了我们来看看效果吧，打开浏览器中输入以下网址：http://localhost/server.asp，然后回车，如果看到如下页面就表示我们的服务发布成功啦。



创建 Hprose 的 Hello 客户端

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<script runat="server" language="JScript">
    Response.CodePage = 65001;
    Response.CharSet = "UTF-8";
    var client = new HproseHttpClient("http://localhost/server.asp");
    Response.write(client.hello("World"));
</script>
```

您可以直接把它复制下来，粘贴到记事本中，然后保存为 client.asp，打开浏览器中输入以下网址：http://localhost/client.asp，如果看到如下输出：

```
Hello World!
```

就说明客户端创建成功了！

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 ASP 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 ASP 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

ASP 里面分为 JScript 和 VBScript 两种脚本，他们分别有不同的类型，它们的类型与 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 JScript/VBScript 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

JScript 类型	VBScript 类型	Hprose 类型
Number 类型中的整数（-2147483648 ~ 2147483647）	Byte，Integer，Long	整型
纯数字字符串	纯数字字符串	长整型
Number 类型中的浮点数	Currency ,Single ,Double	浮点型
NaN	NaN	非数
Infinity	Infinity	正无穷大
-Infinity	-Infinity	负无穷大
true	True	布尔真
false	False	布尔假
undefined，null，function	Empty，Null	空
单字符字符串	单字符 String	字符
字符串	String	字符串型（或空串）
Date 类型	Date（Time）	日期/时间/日期时间型

反序列化类型映射

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	JavaScript 类型	VBScript 类型
整型	Number 类型	Long
长整型	纯数字 String 类型	纯数字 String 类型

Hprose 类型	JavaScript 类型	VBScript 类型
浮点型	Number 类型	Double
非数	Number 类型的 NaN	Double 中的 NaN
正无穷大	Number 类型的 Infinity	Double 中的 Infinity
负无穷大	Number 类型的-Infinity	Double 中的-Infinity
布尔真	true	True
布尔假	false	False
空	null	Null
空串	""	""
二进制型	不支持	不支持
字符/字符串型	String 类型	String
日期/时间/日期时间型	Date 类型	Date

注意：ASP 版本不支持二进制型数据！

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

JScript 和 VBScript 中的 Array 类型数据被映射为 Hprose 列表类型。例如 JScript 中：

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

VBScript 中：

```
Dim a(4)
a(0) = 1
a(1) = 3
a(2) = 5
```

```
   a(3) = 7
   a(4) = 9
```

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 = [];
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 = {};
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

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

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

注意，不要把 Array 类型的对象当 Object 类型使用，例如：

```
var user2 = [];
user2['name'] = 'Tom';
user2['birthday'] = new Date(1982, 2, 23);
user2['age'] = 28;
user2['sex'] = 'Male';
```

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。

VBScript 中可以使用 Directory 类型来表示字典类型。例如：

```
Dim user1
Set user1 = CreateObject("Scripting.Dictionary")
user1.Add "name", "Tom"
user1.Add "birthday", #2/23/1982#
```

```
user1.Add "age", 28
user1.Add "sex", "Male"
```

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

JScript 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。但是我们知道 JScript 并不支持定义类，那么我们怎么样创建自定义类型呢？

JScript 可以通过定义函数的方式来模拟定义类：

```
function User() {}
```

上面这条语句就定义了一个 User 类，下面我们可以这样用它创建一个 User 对象：

```
var user3 = new User();
user3['name'] = 'Tom';
user3['birthday'] = new Date(1982, 2, 23);
user3['age'] = 28;
user3['sex'] = 'Male';
```

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。

但是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 ASP 会自动在接受数据时，创建这个类。但是它所自动创建的类与上面我们定义的并不完全相同，也就是说，Hprose for ASP 提供了另一种自定义可序列化类型的方式。下面我们就来介绍这另外一种方式。

有时候我们并不想定义一个命名函数来创建自定义可序列化类型，例如在使用 Hprose 调试器忘忧草（Nepenthes）的时候，我们只能在一行里写下参数，这时我们就可以采用下面的方式来定义一个自定义可序列化类型对象：

```
var user4 = {'getClassName': function() { return 'User'; },
            'name': 'Tom',
            'birthday': new Date(1982, 2, 23),
            'age': 28,
            'sex': 'Male'};
```

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。

Hprose for ASP 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。

自定义类中的属性名，映射为 Hprose 对象类型中的属性名，自定义类中的属性值，映射为 Hprose 对象类型中的属性值。对于所有的方法属性，序列化时会全部忽略，对于在 Object 对象上，通过 prototype 方式来扩充的属性，在序列化时也会全部忽略。所以 Hprose for ASP 可以很好的跟目前几乎所有的

JavaScript 框架完美结合，而不管这些框架是否对 JavaScript 本身的对象是否有所扩充。

通过 HproseClassManager 来注册自定义类型

JScript 中通过在类名中使用下滑线来定义与其它带有名空间的语言对应的类，例如 JScript 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。

通过命名 function 方式定义的类型，要跟其它语言交互时，需要按照上面的规则来命名 function 才能跟其他语言的定义匹配，通过匿名 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 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 HproseClassManager 的 register 方法就可以轻松实现这个需求。

例如您有一个命名为 User 的 function 类型，希望传递给 C#，C#中与之对应的类是 my.package.User，那么可以这样做：

```
HproseClassManager.register(User, 'my_package_User');
```

这种情况下，您既不需要为 User 添加 getClassName 方法，也不需要更改 function 定义了。

第三章 服务器

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 ASP 服务器，在本章中您将深入的了解 Hprose for ASP 服务器的更多细节。

本章提要

- 发布服务
- 服务器开关
- 服务器事件

发布服务

Hprose 提供了多种方法发布服务，下面我们将对这些方法进行详细介绍。

发布函数

在快速入门一章中，我们已经在 Hello 服务器的例子中看到过如何发布一个函数了，这里我们主要谈一下哪些函数可以作为 Hprose 服务发布。

实际上大部分函数都是可以作为 Hprose 服务发布的。但如果参数或结果中包含有不可序列化类型，那么这种函数就不能够发布。

你可以同时发布多个函数。例如：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseServer.
js"></script>
<script runat="server" language="JScript">
function toLowerCase(str) {
   return str.toLowerCase();
}
function toUpperCase(str) {
   return str.toUpperCase();
}
var hproseServer = new HproseHttpServer();
hproseServer.addFunctions([toLowerCase, toUpperCase]);
hproseServer.handle();
</script>
```

在上面这个例子中我们发布了两个方法。我们可以把方法放到数组中，用 addFunctions 一次发布多个，也可以像快速入门一章中，使用 addFunction 方法一个一个的添加。

大家如果细心的话，会注意到这个例子跟快速入门中的例子还有几点不同：

1. 这个例子是使用 JScript 编写的，而快速入门中的例子是使用 VBScript 编写的。
2. 这个例子中使用的是<script runat=server>标签方式来定义脚本，而快速入门中是使用<% %>方式来定义脚本。
3. 这个例子是使用 new 运算符来创建 HproseHttpServer 对象，而快速入门中是使用 create 方法来创建 HproseHttpServer 对象。
4. 这个例子是使用 handle 方法来启动服务，而快速入门中是使用 start 方法来启动服务。

那么这几点不同有什么需要注意的地方吗？

因为 Hprose for ASP 目前是使用 JScript 编写的，但是根据 ASP 的载入规则，非默认语言的 script 标签先执行，然后<% %>中的内容再执行，默认语言的 script 标签中的内容最后执行。所以不要将默认语言设置为 JScript，否则，将只能使用 JScript 语言，且必须全部使用 script 标签方式。如果保持默认语言为 VBScript，则按照上面例子和快速入门例子中的方式加载就可以正确执行了。

因为 Hprose for ASP 同时支持 JScript 和 VBScript，所以提供了两种创建对象的方法。new 运算符方式用于 JScript 中，这样创建的对象在接收传入的参数时，参数会自动以 JScript 的类型接收。而 create 方法则用于 VBScript 中，这样创建的对象在接受传入的参数时，参数会自动以 VBScript 的类型接收（主要是数组和字典）。

所以最好不要在 VBScript 中发布 JScript 的函数，也不要在 JScript 中发布 VBScript 的函数。

handle 和 start 方法没有任何区别，它们的作用是相同的。

发布方法

Hprose for ASP 也支持发布对象的方法，如下例：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseServer.
js"></script>
<%
   Class MyService
      public function add(a, b)
         add = a + b
      end function
   End Class

   Dim service
   Set service = new MyService

   Dim hproseServer
   Set hproseServer = HproseHttpServer.create()
   hproseServer.addMethod "add", service
   hproseServer.start
%>
```

这里 MyService 是一个 VBScript 定义的类，通过 addMethod 方法将它的 service 对象上的 add 方法发布。

下面是一个 JScript 的例子：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseServer.
js"></script>
<script runat="server" language="JScript">
function MyService() {
   this.foo = function() {
      return 'foo';
   }
   this.bar = function() {
      return 'bar';
   }
```

```
}
var hproseServer = new HproseHttpServer();
hproseServer.addInstanceMethods(new MyService());
hproseServer.start();
</script>
```

通过 addInstanceMethods 你可以很方便的发布对象上定义的所有方法，不过 addInstanceMethods 不适合发布 VBScript 对象上的方法。

现在你可能会有这样的疑问，如果要同时发布两个不同类中的同名方法的话，会不会有冲突呢？如何来避免冲突呢？

别名机制

确实会遇到这种情况，就是当发布的方法同名时，后添加的方法会将前面添加到方法给覆盖掉，在调用时，你永远不可能调用到先添加的同名方法。不过 Hprose 提供了一种别名机制，可以解决这个问题。要用自然语言来解释这个别名机制的话，不如直接看代码示例更直接一些：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseServer.
js"></script>
<script runat="server" language="JScript">
function MyService1() {
    this.foo = function() {
        return 'foo';
    }
    this.bar = function() {
        return 'bar';
    }
}
function MyService2() {
    this.foo = function() {
        return 'foo, too';
    }
    this.bar = function() {
        return 'bar, too';
    }
}var hproseServer = new HproseHttpServer();
hproseServer.addInstanceMethods(new MyService1(), 'ms1');
hproseServer.addInstanceMethods(new MyService2(), 'ms2');
hproseServer.start();
</script>
```

除了向上面这样为一个类中所有的方法提供一个别名前缀以外，还可以给每个方法或者函数提供完整的别名，只需要使用 addFunction、addMethod 或者 addFuntions、addMethods 即可。

最后要注意的一点是，通过别名发布的方法（或函数）在调用时如果用原方法（或函数）名调用是调

用不到的，也就是说只能用别名来调用。

迷失的方法

当客户端调用一个服务器端没有发布的方法时，默认情况下，服务器端会抛出错误。但是如果你希望能对客户端调用的不存在的方法在服务器端做特殊处理的话，你可以通过 addMissingFunction 或 addMissingMethod 方法来实现。

这是一个很有意思的方法，它用来发布一个特定的方法，当客户端调用的方法在服务器发布的方法中没有查找到时，将调用这个特定的方法。

使用 addMissingFunction 可以发布函数，addMissingMethod 可以发布方法，但是只能发布一个。如果多次调用 addMissingFunction 或 addMissingMethod 方法，将只有最后一次发布的有效。

用 addMissingFunction、addMissingMethod 发布的方法参数应该为两个：

第一个参数表示客户端调用时指定的方法名，方法名在传入该方法时全部是小写的。

第二个参数表示客户端调用时传入的参数列表。例如客户端如果传入两个参数，则 args 的列表长度为 2，客户端的第一个参数为 args 的第一个元素，第二个参数为 args 的第二个元素。如果客户端调用的方法没有参数，则 args 为长度为 0 的列表。

除了可直接使用 addMissingFunction、addMissingMethod 来处理迷失的方法以外，你还可以通过 addFunction 或 addMethod 发布一个别名为星号（*）的方法。效果是一样的。

服务器开关

隐藏发布列表

发布列表的作用相当于 Web Service 的 WSDL，与 WSDL 不同的是，Hprose 的发布列表仅包含方法名，而不包含方法参数列表，返回结果类型，调用接口描述，数据类型描述等信息。这是因为 Hprose 是支持弱类型动态语言调用的，因此参数个数，参数类型，结果类型在发布期是不确定的，在调用期才会确定。所以，Hprose 与 Web Service 相比无论是服务的发布还是客户端的调用都更加灵活。

如果您不希望用户直接通过浏览器就可以查看发布列表的话，您可以禁止服务器接收 GET 请求。方法很简单，只需要在调用 handle 方法之前调用 setGetEnabled 方法，将参数设置为 false 即可。

好了，现在通过 GET 方式访问不再显示发布列表啦。但是客户端调用仍然可以正常执行，丝毫不受影响。不过在调试期间，不建议禁用发布列表，否则将会给您的调试带来很大的麻烦。也许您更希望能够在调试期得到更多的调试信息，那这个可以做到吗？答案是肯定的，您只要打开调试开关就可以了。

调试开关

默认情况下，在调用过程中，服务器端发生错误时，只返回有限的错误信息。当打开调试开关后，服务器会将错误堆栈信息全部发送给客户端，这样，您在客户端就可以看到详细的错误信息啦。

开启方法很简单，只需要在调用 handle 方法之前调用 setDebugEnabled 方法，将参数设置为 true 即可。

该开关目前在 ASP 版本中仅为兼容其它语言版本而保留，暂时不起作用。

P3P 开关

在 Hprose 的 http 服务中还有一个 P3P 开关，这个开关决定是否发送 P3P 的 http 头，这个头的作用是让 IE 允许跨域接收的 Cookie。当您的服务需要在浏览器中被跨域调用，并且希望传递 Cookie 时（例如通过 Cookie 来传递 Session ID），您可以考虑将这个开关打开。否则，无需开启此开关。此开关默认是关闭状态。开启方法与上面的开关类似，只需要在调用 handle 方法之前调用 setP3PEnabled 方法，将参数设置为 true 即可。

跨域开关

Hprose 支持 JavaScript、ActionScript 和 SilverLight 客户端的跨域调用，对于 JavaScript 客户端来说，服务器提供了两种跨域方案，一种是 W3C 标准跨域方案，这个只需要在服务器端调用 handle 方法之前调用 setCrossDomainEnabled 方法，将参数设置为 true 即可。当你在使用 Hprose 专业版提供的服务测试工具 Nepenthes（忘忧草）时，一定要注意必须要打开此开关才能正确进行调试，否则 Nepenthes 将报告错误的服务器。

另一种跨域方案同时适用于以上三种客户端，那就是通过设置跨域策略文件的方式。这个只需要将 crossdomain.xml 放在服务器发布的根目录上即可。

对于 SilverLight 客户端来说，还支持 clientaccesspolicy.xml 这个客户端访问策略文件，它的设置方法跟 crossdomain.xml 是一样的，都是放在服务器发布的根目录上。

关于 crossdomain.xml 和 clientaccesspolicy.xml 的更多内容请参阅：

- http://www.adobe.com/devnet/articles/crossdomain_policy_file_spec.html
- http://www.adobe.com/devnet/flashplayer/articles/fplayer9_security.html
- http://msdn.microsoft.com/en-us/library/cc197955%28v=VS.95%29.aspx
- http://msdn.microsoft.com/en-us/library/cc838250%28v=VS.95%29.aspx

服务器事件

也许您可能还希望设置其它的 http 头，或者希望在发生错误时，能够在服务器端进行日志记录。甚至希望在调用发生的前后可以做一些权限检查或日志记录等。在 Hprose 中，这些都可以轻松做到。Hprose 提供了这样的事件机制。

Hprose 服务器提供了四个事件，它们分别是 onBeforeInvoke、onAfterInvoke、onSendHeader 和 onSendError。下面我们就来对这四个事件分别做一下介绍。

onBeforeInvoke 事件

当服务器端发布的方法被调用前，onBeforeInvoke 事件被触发，它有三个参数，他们从左到右的顺序分别是 name，args 和 byRef。其中 name 为客户端所调用的方法名，args 为方法的参数，byRef 表示是否是引用参数传递的调用。

您可以在该事件中做用户身份验证，例如 IP 验证。也可以作日志记录。如果在该事件中想终止调用，抛出异常即可。

onAfterInvoke 事件

当服务器端发布的方法被成功调用后 ,onAfterInvoke 事件被触发 ,其中前三个参数与 onBeforeInvoke 事件一致，最后一个参数 result 表示调用结果。

当调用发生错误时，onAfterInvoke 事件将不会被触发。如果在该事件中抛出异常，则调用结果不会被返回，客户端将收到此事件抛出的异常。

onSendHeader 事件

当服务器返回响应头部时，onSendHeader 事件会被触发，该事件无参数。

在该事件中，您可以发送您自己的头信息，例如设置 Cookie。该事件中不应抛出任何异常。

onSendError 事件

当服务器端调用发生错误，或者在 onBeforeInvoke、onAfterInvoke 事件中抛出异常时，该事件被触发，该事件只有一个参数 error。

您可以在该事件中作日志记录，但该事件中不应再抛出任何异常。

第四章 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 ASP 客户端，在本章中您将深入的了解 Hprose for ASP 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
- 保持会话

同步调用

直接通过远程方法名进行远程调用

在快速入门一章中，我们已经见识过这种方式的调用了，这里再来具一个例子来进行说明：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<%
    Response.CodePage = 65001
    Response.addHeader "ContentType", "text/html; charset=UTF-8"
    Set client = HproseHttpClient.create("http://www.hprose.com/example/")
    Response.write(client.sum(1, 2, 3, 4, 5))
    Response.write("<br />")
    Response.write(client.sum(6.0, 7.0, 8.0))
    Response.write("<br />")
    Dim users, i
    users = client.getUserList()
    for i = 0 to 3
        Response.write("Name:" & users(i).name & " ")
        Response.write("Sex:" & users(i).sex & " ")
        Response.write("Birthday:" & users(i).birthday & " ")
        Response.write("Age:" & users(i).age & " ")
        Response.write("Married:" & users(i).married & " ")
        Response.write("<br />")
    next
%>
```

上面的主程序代码是使用 VBScript 编写的 ,创建客户端对象使用的是 HproseHttpClient 的 create 方法，使用该方法创建的对象 ,在进行远程调用时 ,返回的数据中如果包含有数组 ,则该数组会自动转换为 VBScript 的数组，如果包含有字典，则会以 Scripting.Dictionary 对象形式返回。如果返回结果包含有自定义对象的话，则以 JScript 中的自定义对象返回。

上面程序的运行结果为：

```
15
21
Name:Amy Sex:2 Birthday:1983-12-3 Age:26 Married:True
Name:Bob Sex:1 Birthday:1989-6-12 Age:20 Married:False
Name:Chris Sex:0 Birthday:1980-2-29 Age:29 Married:True
Name:Alex Sex:3 Birthday:1992-6-14 Age:17 Married:False
```

如果代码采用 JScript 编写，则创建客户端对象时，应该使用 new 运算符来创建 HproseHttpClient 对

象 ,这样创建的对象 ,在进行远程调用时 ,返回的结果全部是 JScript 的类型。例如上面的例子如果使用 JScript 编写，则代码如下：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<script runat="server" language="JScript">
    Response.CodePage = 65001;
    Response.addHeader("ContentType", "text/html; charset=UTF-8");
    var client = new HproseHttpClient("http://www.hprose.com/example/");
    Response.write(client.sum(1, 2, 3, 4, 5));
    Response.write("<br />");
    Response.write(client.sum(6.0, 7.0, 8.0));
    Response.write("<br />");
    var users = client.getUserList();
    for (var i = 0; i < 4; i++) {
        Response.write("Name:" + users[i].name + " ");
        Response.write("Sex:" + users[i].sex + " ");
        Response.write("Birthday:" + users[i].birthday + " ");
        Response.write("Age:" + users[i].age + " ");
        Response.write("Married:" + users[i].married + " ");
        Response.write("<br />");
    }
</script>
```

我们会发现，调用结果是一样的，只是在输出结果的形式上略有不同。

通过 Invoke 方法进行远程调用

上面介绍的通过远程方法名进行远程调用的例子用 invoke 方法重写的代码：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<%
    Response.CodePage = 65001
    Response.addHeader "ContentType", "text/html; charset=UTF-8"
    Set client = HproseHttpClient.create("http://www.hprose.com/example/")
    Response.write(client.invoke("sum", 1, 2, 3, 4, 5))
    Response.write("<br />")
    Response.write(client.invoke("sum", 6.0, 7.0, 8.0))
    Response.write("<br />")
    Dim users, i
    users = client.invoke("getUserList")
    for i = 0 to 3
```

```
        Response.write("Name:" & users(i).name & " ")
        Response.write("Sex:" & users(i).sex & " ")
        Response.write("Birthday:" & users(i).birthday & " ")
        Response.write("Age:" & users(i).age & " ")
        Response.write("Married:" & users(i).married & " ")
        Response.write("<br />")
    next
%>
```

运行结果与上面例子的运行结果完全相同。

异步调用

直接通过远程方法名进行远程调用

目前异步调用仅支持使用 JScript 编写的客户端，上面同步调用的例子改为异步调用后的代码如下：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<script runat="server" language="JScript">
    function printSum(result) {
        Response.write(result);
        Response.write("<br />");
    }
    Response.CodePage = 65001;
    Response.addHeader("ContentType", "text/html; charset=UTF-8");
    var client = new HproseHttpClient("http://www.hprose.com/example/");
    client.sum_callback = printSum;
    client.sum(1, 2, 3, 4, 5);
    client.sum(6.0, 7.0, 8.0);
    client.getUserList(function(users) {
        for (var i = 0; i < 4; i++) {
            Response.write("Name:" + users[i].name + " ");
            Response.write("Sex:" + users[i].sex + " ");
            Response.write("Birthday:" + users[i].birthday + " ");
            Response.write("Age:" + users[i].age + " ");
            Response.write("Married:" + users[i].married + " ");
            Response.write("<br />");
        }
    });
    client.waitForResponse();
</script>
```

这个例子的运行结果如下：

```
15
Name:Amy Sex:2 Birthday:Sat Dec 3 00:00:00 UTC+0800 1983 Age:26 Married:true
Name:Bob Sex:1 Birthday:Mon Jun 12 00:00:00 UTC+0800 1989 Age:20 Married:false
Name:Chris Sex:0 Birthday:Fri Feb 29 00:00:00 UTC+0800 1980 Age:29 Married:true
Name:Alex Sex:3 Birthday:Sun Jun 14 00:00:00 UTC+0800 1992 Age:17 Married:false
21
```

因为是异步调用，所以运行结果的输出顺序是不一定的。异步调用可以使用事件方式，也可以使用回调方式。事件方式中，事件名称为“远程方法名_callback”或者“远程方法名_onSuccess”。而回调方式则采用直接在远程调用中将回调函数作为最后一个参数传入。

另外，在 ASP 客户端中，我们提供了一个 waitForResponse 方法，当调用这个方法后，则直到所有的异步调用全部执行结束后再执行后面的语句。

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：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<script runat="server" language="JScript">
   Response.CodePage = 65001;
   Response.addHeader("ContentType", "text/html; charset=UTF-8");
   var client = new HproseHttpClient("http://www.hprose.com/example/");
   var arg = {"Mon": 1, "Tue": 2, "Wed": 3, "Thu": 4, "Fri": 5, "Sat": 6, "Sun": 7};
   client.swapKeyAndValue(arg, function(result, args) {
      for (var k in result) { Response.write(k + ":" + result[k] + "; "); }
      Response.write("<br />");
      for (var k in args[0]) { Response.write(k + ":" + args[0][k] + "; "); }
      Response.write("<br />");
      for (var k in arg) { Response.write(k + ":" + arg[k] + "; "); }
      Response.write("<br />");
   }, true);
   client.waitForResponse();
</script>
```

上面程序的运行结果如下：

```
1:Mon; 2:Tue; 3:Wed; 4:Thu; 5:Fri; 6:Sat; 7:Sun;
1:Mon; 2:Tue; 3:Wed; 4:Thu; 5:Fri; 6:Sat; 7:Sun;
Mon:1; Tue:2; Wed:3; Thu:4; Fri:5; Sat:6; Sun:7;
```

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 true，这个参数指定该调用为引用参数传递。

Hprose 1.2 for ASP 中，客户端对象上增加了 setByRef 和 getByRef 方法，您可以通过它来设置或获取是否为引用参数传递的默认值。但通常您不需要调用它，除非您所有的调用都是引用参数传递。

通过服务代理对象进行调用

所谓服务代理对象即：它作为远程服务的本地代理，包含了可以直接调用的远程方法，在它上面可以像调用本地方法那样去调用远程服务。您可以把 HproseHttpClient 对象看成是一个特殊的服务代理对象，例如上面所讲的用法都是将 HproseHttpClient 对象作为远程服务代理对象使用的。

但是如果服务器端发布的方法一旦与 HproseHttpClient 本身包含的方法名一致的话，这时如果还将 HproseHttpClient 对象作为远程服务代理对象来使用的话，就会发生冲突，您将无法调用到远程方法。

这时，您就需要使用单独的远程服务代理对象来代替 HproseHttpClient 对象了。

Hprose 1.2 for ASP 及其之后的版本提供了这样的功能。您只需要在创建 HproseHttpClient 时，不指定任何参数，之后用 useService 来指定相应参数即可返回单独的远程服务代理对象，例如：

```
   var client = new HproseHttpClient();
   var serviceProxy = client.useService("http://www.hprose.com/example/", ['hello'], true)
;
   serviceProxy.hello("World!", function(result) {  Response.write(result); });
   client.waitForResponse();
```

这里 useService 的第三个参数很重要 ,它为 true 才表示创建单独的远程服务代理对象 ,否则会将 client 本身作为远程服务代理对象。

另外，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，而有了远程服务代理对象之后，您可以让某个别名前缀直接变成一个远程服务代理对象，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：

```
var client = new HproseHttpClient(url, ['exam1_login', 'exam1_add', 'exam1_update', 'exam1_
delete']);
client.exam1_login('admin', '123456', function(result) {
   Response.write(result);
}
client.exam1_add(user, function(result) {
   Response.write(result);
}
client.exam1_update(12, user2, function(result) {
```

```
    Response.write(result);
}
client.exam1_delete(8, function(result) {
    Response.write(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远程服务代理对象之后怎样写：

```
var client = new HproseHttpClient(url, {'exam1': ['login', 'add', 'update', 'delete']});
var exam1 = client.exam1;
exam1.login('admin', '123456', function(result) {
    Response.write(result);
}
exam1.add(user, function(result) {
    Response.write(result);
}
exam1.update(12, user2, function(result) {
    Response.write(result);
}
exam1.delete(8, function(result) {
    Response.write(result);
}
```

您会发现，client.exam1 直接作为一个远程服务代理对象返回了，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。

如果您有多个远程对象，还可以这样写：

```
var client = new HproseHttpClient(url, [
    {'exam1': ['login', 'add', 'update', 'delete']},
    {'exam2': ['login', 'add', 'update', 'delete']},
]);
```

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：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
js"></script>
<script runat="server" language="JScript">
    function MyUser() {
        this.name = 'Tom';
        this.birthday = new Date(1982, 2, 23);
        this.age = 28;
```

```
        this.sex = 1;
        this.married = true;
    }
    HproseClassManager.register(MyUser, 'User');
    var client = new HproseHttpClient("http://www.hprose.com/example/", {'print':'r'});
    Response.write(client.print.r(new MyUser(), true));
</script>
```

让我们来看看运行结果吧：

```
User Object
(
    [name] => Tom
    [sex] => 1
    [birthday] => HproseDate Object
        (
            [year] => 1982
            [month] => 3
            [day] => 23
            [utc] =>
        )

    [age] => 28
    [married] => 1
)
```

非常棒！我们成功了！

上面这个例子是同步调用，异步调用的用法是完全相同的。也就是说代理对象方式不仅仅在异步调用时可以使用，在同步调用时也同样可以使用。

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，这就是远程服务代理对象的用法。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。

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 ASP 还提供了通过 invoke 方法进行远程调用。这通常在方法未知，或者动态调用时用到。下面是用 invoke 方法重写的代码：

```
<%@ CODEPAGE=65001 %>
<script runat="server" language="JScript" type="text/javascript" src="hprose/hproseClient.
```

```
js"></script>
<script runat="server" language="JScript">
    function printSum(result) {
        Response.write(result);
        Response.write("<br />");
    }
    Response.CodePage = 65001;
    Response.addHeader("ContentType", "text/html; charset=UTF-8");
    var client = new HproseHttpClient("http://www.hprose.com/example/", []);
    client.sum_callback = printSum;
    client.invoke("sum", 1, 2, 3, 4, 5);
    client.invoke("sum", 6.0, 7.0, 8.0);
    client.invoke("getUserList", function(users) {
        for (var i = 0; i < 4; i++) {
            Response.write("Name:" + users[i].name + " ");
            Response.write("Sex:" + users[i].sex + " ");
            Response.write("Birthday:" + users[i].birthday + " ");
            Response.write("Age:" + users[i].age + " ");
            Response.write("Married:" + users[i].married + " ");
            Response.write("<br />");
        }
    });
</script>
```

运行结果与直接通过方法名进行远程调用的第一个例子的运行结果完全相同。我们发现除了换成了 invoke 之外，我们还在初始化 HproseHttpClient 对象时指定了一个空的远程方法列表，这样做是为了避免多余的跟服务器之间的通讯。

引用参数传递

引用参数传递类似：

```
    var client = new HproseHttpClient("http://www.hprose.com/example/", []);
    var arg = {"Mon": 1, "Tue": 2, "Wed": 3, "Thu": 4, "Fri": 5, "Sat": 6, "Sun": 7};
    client.invoke("swapKeyAndValue", arg, function(result, args) {
        for (var k in result) { Response.write(k + ":" + result[k] + "; "); }
        Response.write("<br />");
        for (var k in args[0]) { Response.write(k + ":" + args[0][k] + "; "); }
        Response.write("<br />");
        for (var k in arg) { Response.write(k + ":" + arg[k] + "; "); }
        Response.write("<br />");
    }, true);
    client.waitForResponse();
```

运行结果与直接通过方法名进行远程调用的引用参数传递例子的运行结果完全相同。不再多做解释。

异常处理

Hprose for ASP 支持同步调用和异步调用。

当在同步调用中发生异常时，可以通过 try…catch 来捕获异常。

当在异步调用中发生异常时，异常将不会被抛出。如果您希望能够处理这些异常，只需要给 Hprose 客户端对象指定合适的 onError 事件即可，onError 事件有两个参数，使用非常简单，例如：

```
var client = new HproseHttpClient("http://www.hprose.com/example/", ['Hi']);
client.onError = function(name, error) {
    Response.write(name);
    Response.write(error.message);
}
client.Hi("Hprose", function(result) {
    Response.write(result);
});
client.waitForResponse();
```

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：

```
HiCan't find this function Hi().
file: D:\phpox\hprose\www\example\hproseHttpServer.php
line: 166
trace: #0 D:\phpox\hprose\www\example\hproseHttpServer.php(396): HproseHttpServer->doInvok
e()
#1 D:\phpox\hprose\www\example\hproseHttpServer.php(413): HproseHttpServer->handle()
#2 D:\phpox\hprose\www\example\index.php(60): HproseHttpServer->start()
#3 {main}
```

在 Hprose 1.2 for ASP 及其之后的版本中，您还可以直接将错误处理函数作为参数带入调用中，这样可以给每个调用指定单独的错误处理方式。例如上面的例子还可以写做：

```
var client = new HproseHttpClient("http://www.hprose.com/example/", ['Hi']);
client.Hi("Hprose", function(result) {
    Response.write(result);
},
function(name, error) {
    Response.write(name);
    Response.write(error.message);
});
client.waitForResponse();
```

超时设置

Hprose 1.2 for ASP 及其之后的版本中增加了超时设置。只需要调用客户端对象上的 setTimeout 方法即可，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。

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，例如代理服务器、http 标头等设置，下面我们来分别介绍。

代理服务器

默认情况下，代理服务器是被禁用的。可以通过 setProxy 方法来设置 http 代理服务器的地址和端口。参数是代理服务器的完整地址字符串，例如："http://10.54.1.39:8000"。另外，还可以设置代理验证所使用的用户名和密码。

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 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。

保持会话

当浏览器访问一个 ASP 页面，而这个 ASP 页面又通过 Hprose 客户端去访问另一台 Hprose 服务器时，通常是不能保持浏览器到 Hprose 客户端访问的那台 Hprose 服务器会话的。那有什么办法能够将这个会话保持并传递到浏览器吗？

Hprose 1.2 for ASP 中提供了这样的功能，只需要在使用客户端页面的开头调用：

```
HproseHttpClient.keepSession();
```

就可以了。

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 ASP 中，序列化的结果以 String 类型返回。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 枚举中定义。

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。