



# 用户手册

1.3

（JavaScript 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<br>产品购买及项目定制： sales@hprfc.com<br>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 JavaScript 进行分布式程序开发的全过程。

## 本章提要

- 安装 Hprose for JavaScript
- 创建 Hprose 的 Hello 客户端

# 安装 Hprose for JavaScript

Hprose for JavaScript 支持 IE5.5+、Firefox、Safari、Chrome、Opera、Camino、SeaMonkey 等各种现代浏览器，以及这些浏览器的移动版本。

Hprose for JavaScript 支持 AIR 开发。

Hprose for JavaScript 支持 SilverLight 1.0。

Hprose for JavaScript 支持 WebOS 开发。

## 安装方法

Hprose for JavaScript 包含源码版本和压缩版本这两个版本。其中 hproseCommon.js，hproseIO.js，hproseHttpRequest.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，而应该使用压缩版本的 hprose.js。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 type="text/javascript" src="hproseCommon.js"></script>
<script type="text/javascript" src="hproseIO.js"></script>
<script type="text/javascript" src="hproseHttpRequest.js"></script>
<script type="text/javascript" src="hproseHttpClient.js"></script>
```

# 创建 Hprose 的 Hello 客户端

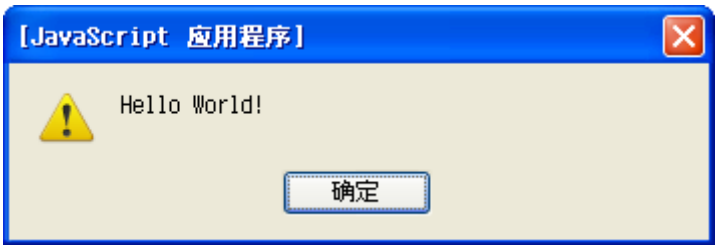
Hprose for JavaScript 的 1.0 版本只提供了客户端功能，因此我们还需要一个服务器才能进行讲解和演示。不过这并不是什么问题，Hprose 的客户端和服务器所使用的语言是没有必然联系的，不论服务器是 Java、C#还是 PHP，对客户端来说都是一样的。所以这里我们将使用 PHP 版本的 Hprose 服务器为例来进行讲解。为了方便大家可以直接使用本书中的例子进行练习，我们直接将服务器放在了官方网站上，地址为：http://www.hprose.com/example/，这是一个实际可用的服务，后面所有的例子，都以这个地址所提供的服务为例来进行说明。

开发 Hprose for JavaScript 程序您并不需要什么特殊的编辑器或 IDE，只要有 Windows 自带的记事本或者 Linux 下自带的 vi 就可以开始了。下面是完整的 Hello World 客户端实例代码：

```
<html>
<head>
   <script type="text/javascript" src="hprose.js"></script>
</head>
<body>
   <script type="text/javascript">
       var client = new HproseHttpClient("http://www.hprose.com/example/", ["hello"]);
        client.hello("World!", function(result) {
           alert(result);
        });
    </script>
```

```
</body>
</html>
```

您可以直接把它复制下来，粘贴到记事本中，然后保存为 hello.html，并保证跟压缩版本的 hprose.js 在同一个目录下，之后双击用浏览器打开它，您就可以看到运行结果了：




如果您看到上面的画面，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 JavaScript 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 JavaScript 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

JavaScript 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 JavaScript 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

| JavaScript 类型 | Hprose 类型 |
|---|---|
| Number 类型中的整数（-2147483648 ~ 2147483647） | 整型 |
| 纯数字字符串 | 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，function | 空 |
| 单字符字符串 | 字符 |
| 字符串 | 字符串型（或空串） |
| Date 类型 | 日期/时间/日期时间型 |

## 反序列化类型映射

| Hprose 类型 | JavaScript 类型 |
|---|---|
| 整型 | Number 类型 |
| 长整型 | 纯数字 String 类型 |
| 浮点型 | Number 类型 |
| 非数 | Number 类型的 NaN |

| Hprose 类型 | JavaScript 类型 |
| --- | --- |
| 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 | 不支持 |
| 字符/字符串型 | String 类型 |
| 日期/时间/日期时间型 | Date 类型 |

注意：JavaScript 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

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

```
var a1 = new Array(1, 2, 3, 4, 5);
var a2 = new Array(3);
var a3 = ['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 = [];
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

JavaScript 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。

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

JavaScript 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 JavaScript 并不支持定义类，那么我们怎么样创建自定义类型呢？

JavaScript 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 JavaScript 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 JavaScript 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 JavaScript 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 JavaScript 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 本身的对象有所扩充。

## 通过 HproseClassManager 来注册自定义类型

JavaScript 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 JavaScript 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？

在 Hprose 1.2 for JavaScript 中，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 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 JavaScript 客户端，在本章中您将深入的了解 Hprose for JavaScript 客户端的更多细节。

## 本章提要

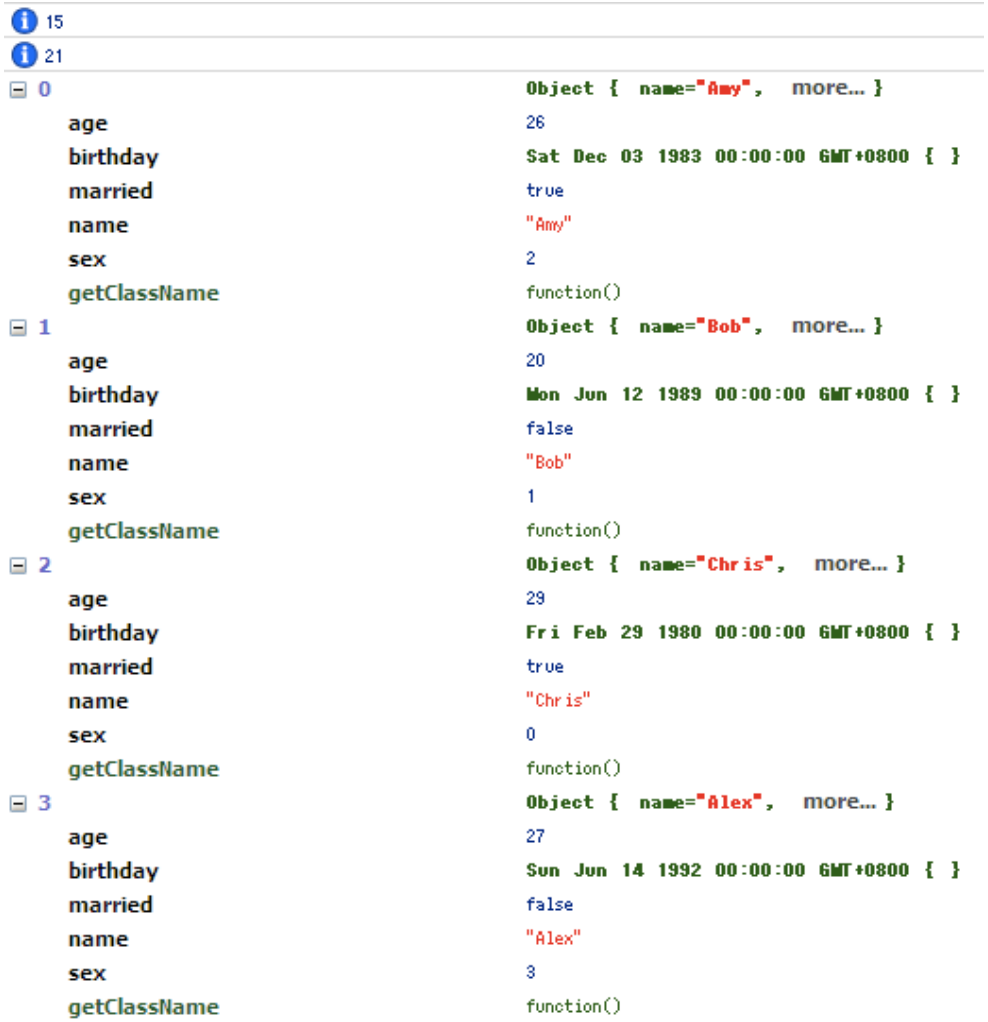
- 直接通过远程方法名进行远程调用
- 通过 Invoke 方法进行远程调用
- 异常处理
- 超时设置
- HTTP 标头设置
- 跨域调用

# 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 ,下面的例子我们为了方便讲解 ,使用了 Firebug 中的调试方法 ,所以下面的例子需要您在 Firefox 中安装 Firebug 插件才能运行。当然您也可以使用 Firebug lite 在其它浏览器中运行。请看实例：

```
var client = new HproseHttpClient("http://www.hprose.com/example/", ["sum", "getUserList"]
);
client.sum(1, 2, 3, 4, 5, function(result) {
    console.info(result);
});
client.sum(6.0, 7.0, 8.0, function(result) {
    console.info(result);
});
client.getUserList(function(result) {
    console.dir(result);
});
```

这个例子的运行结果如下：



## 异步顺序调用

Hprose for JavaScript 客户端只有异步调用方式，上面这个结果虽然是按照顺序显示的，但如果您多次执行，您会发现这个结果顺序并不是固定的。如果想要固定顺序的返回结果，可以用下面的方式：

```
client.sum(1, 2, 3, 4, 5, function(result) {
   console.info(result);
   client.sum(6.0, 7.0, 8.0, function(result) {
      console.info(result);
      client.getUserList(function(result) {
         console.dir(result);
      });
   });
});
```

从这个例子中，我们可以看到如果远程方法返回结果中包含有某个类的对象，而该类并没有在客户端明确定义的话，Hprose 会自动帮您生成这个类的定义（例如上例中的 User 类），并返回这个类的对象。

上面例子中，在初始化 HproseHttpClient 时，除了指定服务器地址以外，还要指定服务器方法列表，这样才可以直接通过方法名进行调用，方法名大小写跟服务器方法列表中指定的名称要完全一致才行，因为 JavaScript 是区分大小写的。

## onReady 事件

那么是否可以不指定服务器方法列表呢？可以，但是不推荐，因为当不指定服务器方法列表时，客户端会自动去服务器端获取方法列表，这样会增加一次跟服务器的通讯，另外，因为这个获取列表的动作也是异步的，所以，如果要进行调用的话，需要在 onReady 事件中才能够保证调用成功，下面是不指定服务器方法列表的写法：

```
var client = new HproseHttpClient("http://www.hprose.com/example/");
client.onReady = function() {
   client.sum(1, 2, 3, 4, 5, function(result) {
      console.info(result);
   });
   client.sum(6.0, 7.0, 8.0, function(result) {
      console.info(result);
   });
   client.getUserList(function(result) {
      console.dir(result);
   });
};
```

## 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：

```
var client = new HproseHttpClient("http://www.hprose.com/example/", ["swapKeyAndValue"]);
var arg = {"Mon": 1, "Tue": 2, "Wed": 3, "Thu": 4, "Fri": 5, "Sat": 6, "Sun": 7};
client.swapKeyAndValue(arg, function(result, args) {
    console.dir(arg);
    console.dir(args);
    console.dir(result);
}, true);
```

上面程序的运行结果如下：


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 JavaScript 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 JavaScript 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) {  alert(result); });
```

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) {
   alert(result);
}
client.exam1_add(user, function(result) {
   alert(result);
}
client.exam1_update(12, user2, function(result) {
   alert(result);
}
client.exam1_delete(8, function(result) {
   alert(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) {
   alert(result);
}
exam1.add(user, function(result) {
   alert(result);
}
```

```
exam1.update(12, user2, function(result) {
   alert(result);
}
exam1.delete(8, function(result) {
   alert(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。

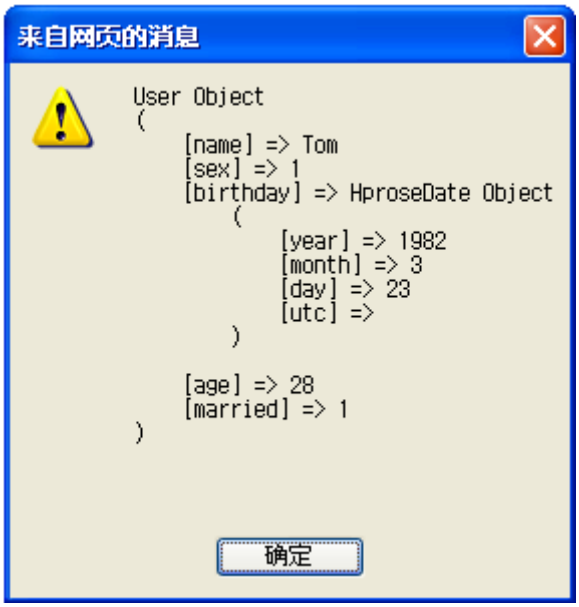
如果您有多个远程对象，还可以这样写：

```
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：

```
<html>
<head>
   <script type="text/javascript" src="hprose.js"></script>
</head>
<body>
   <script type="text/javascript">
      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'});
        client.print.r(new MyUser(), true, function(result) {
            alert(result);
        });
    </script>
</body>
</html>
```

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 JavaScript 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：

```
var client = new HproseHttpClient("http://www.hprose.com/example/", []);
client.invoke("sum", 1, 2, 3, 4, 5, function(result) {
    console.info(result);
});
client.invoke("sum", 6.0, 7.0, 8.0, function(result) {
    console.info(result);
});
client.invoke("getUserList", function(result) {
    console.dir(result);
});
```

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) {
    console.dir(arg);
    console.dir(args);
    console.dir(result);
}, true);
```

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 JavaScript 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。

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) {
    console.info(name);
    console.info(error);
}
client.Hi("Hprose", function(result) {
    console.info(result);
});
```

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：


在 Hprose 1.2 for JavaScript 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) {
    console.info(result);
}, function(name, error) {
    console.info(name);
```

```
    console.info(error);
});
```

# 超时设置

Hprose 1.2 for JavaScript 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 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 JavaScript 最激动人心的功能就是提供了跨域调用，而且提供了两种方式，一种方式是通过 W3C 标准的跨域权限控制方式，另一种是通过 FlashRequest 跨域方式。下面我们来分别介绍。

## W3C 标准跨域权限控制

W3C 标准跨域权限控制支持的浏览器有 IE 8、Firefox 3.5、Safari 4.0、Chrome 3.0、iOS Safari 3.2、Android Browser 2.1 及其这些浏览器的更新版本。Opera 目前还没有版本支持这个功能。

这种方式下，Hprose for JavaScript 客户端不需要做任何特别设置，只需要在 Hprose 服务器端开启 CrossDomainEnabled 属性即可。

W3C 标准跨域标准的英文为：Cross-Origin Resource Sharing，简写为 cors，在 Hprose 1.2 for JavaScript 中，可以通过 HproseHttpClient.corsSupport 来判断当前浏览器是否支持 W3C 的跨域标准。当判断为不支持时，可以选择使用下面这种跨域方式。

## FlashRequest 跨域方式

Hprose for JavaScript 还提供了 FlashRequst 跨域方式，它的原理是借助 Flash 来提交 Hprose 请求，而不借助 XmlHttpRequest 来跟服务器通讯。因此这种方式下，需要浏览器必须安装有 Flash Player 9 以上版本的插件。另外，还需要在脚本中加入下面一句来开启 FlashRequest 功能：

```
HproseHttpClient.setFlash();
```

如果您的 hproseHttpRequest.swf 文件并不在与程序相同的目录下的话，您还可以通过这样指定路径：

```
HproseHttpClient.setFlash('js/hprose/');
```

上面的路径指定了相对于当前目录下的 js 目录下的 hprose 目录下的 hproseHttpRequest.swf 文件。注意这里的路径最后一定要包含“/”，但一定不要包含 hproseHttpRequest.swf 文件名。

注意，上面这句脚本，一定要放在 html 的 body 部分，而不能放在 head 部分中。

除了客户需要上面这个简单设置之外，服务器端也需要对跨域权限进行一下简单的设置。设置方法与 Flash 跨域请求权限设置相同，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 JavaScript 中，序列化的结果以字符串方式表示。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 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 JavaScript 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。