用户手册

1.3

（PHP 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 PHP 进行分布式程序开发的全过程。

本章提要

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

安装 Hprose for PHP

Hprose for PHP 支持 PHP 5.0 或更高版本，并且支持 SAE 云计算平台。

如果需要运行服务器，那么可以选择 Apache、IIS、lighttpd、nginx 或其它任何一款可以运行 PHP 的 Web 服务器。在 SAE 云计算平台上你不需要单独的服务器。

安装方法

Hprose for PHP 有三个版本 ,分别是面向普通的 PHP 配置 ,带有 curl 扩展的 PHP 配置和 SAE 平台的。这三个版本都包含有 hproseCommon.php ， hproseIO.php ， hproseHttpServer.php 和 hproseHttpClient.php 这四个文件，目前只有 hproseHttpClient.php 在这三个版本中是有区别的，但是用法完全相同。使用时，直接复制到您的开发环境的目录下即可，无需任何安装步骤。

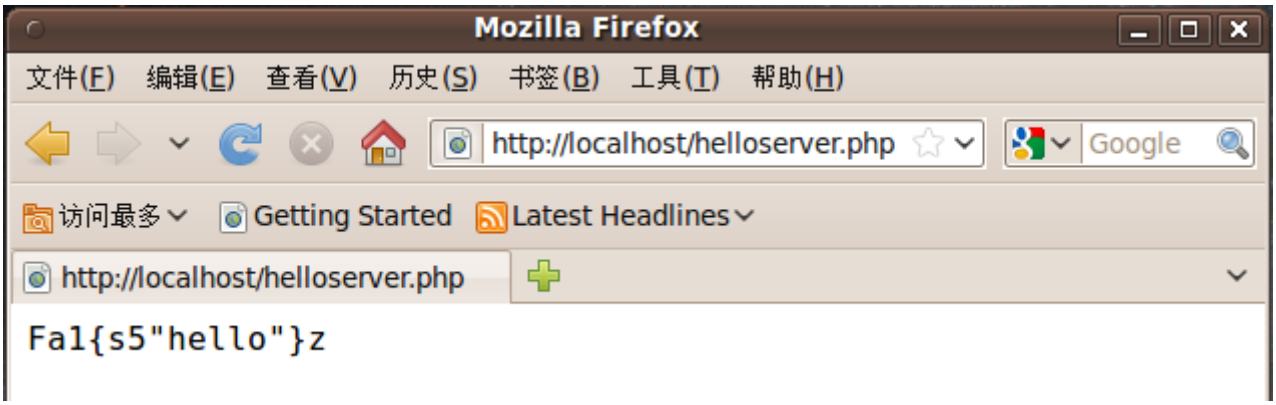
创建 Hprose 的 Hello 服务器

创建 PHP 的 Hprose 的服务器非常简单。下面我们以 Linux 环境为例来讲解。假设在 Linux 下已经配置好了 Web 服务器（apache、lighttpd 或其它任何 http 服务器），并且可以执行 PHP 程序，发布路径为 /var/www。然后我们在发布目录下建立一个 hprose 目录，把 Hprose for PHP 的几个文件放到该目录下。然后在发布目录下创建一个 helloserver.php 文件，内容如下：

```
<?php
include("hprose/hproseHttpServer.php");
function hello($name) {
    return "Hello " . $name;
}
$server = new HproseHttpServer();
$server->addFunction("hello");
$server->handle();
?>
```

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/helloserver.php，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。



接下来我们来看一下客户端如何创建吧。

创建 Hprose 的 Hello 客户端

PHP 的 Hprose 客户端创建更加容易，同样在发布目录下创建 helloclient.php 文件，内容如下：

```
<?php
include("hprose/hproseHttpClient.php");
$client = new HproseHttpClient("http://localhost/helloserver.php");
echo $client->Hello("Hprose");
?>
```

然后打开浏览器输入 http://localhost/helloclient.php，或者在命令行上输入 php helloclient.php，如果看到下面的结果：

```
Hello Hprose
```

就说明客户端创建成功了！

到这里，您应该已经掌握 Hprose for PHP 的基本用法啦。接下来，就让我们一起对 Hprose for PHP 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 PHP 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

PHP 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 PHP 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

PHP 类型	Hprose 类型
整数	整型
纯数字字符串	长整型
浮点数	浮点型
is_nan 为 true 的浮点数	非数
is_infinite 为 true 且大于 0 的浮点数	正无穷大
is_infinite 为 true 且小于 0 的浮点数	负无穷大
true	布尔真
false	布尔假
NULL	空
非 utf8 字符串	二进制型（或空串）
单字元 utf8 字符	字符
utf8 字符串	字符串型（或空串）
HproseDate 对象	日期型
HproseTime 对象	时间型
HproseDateTime 对象	日期时间型

反序列化类型映射

Hprose 类型	PHP 类型
整型	整数

Hprose 类型	PHP 类型
长整型	纯数字字符串
浮点型	浮点数
非数	浮点数中的 NaN
正无穷大	浮点数中的正无穷大
负无穷大	浮点数中的负无穷大
布尔真	true
布尔假	false
空	NULL
空串	""
二进制型	字符串
字符/字符串型	utf8 编码的字符串
日期型	HproseDate 对象
时间型	HproseTime 对象
日期时间型	HproseDateTime 对象

容器类型

Hprose 中的容器类型包括列表类型和字典类型两种。它们都对应于 PHP 的数组类型。

列表类型

任何以从 0 开始的连续整数作为索引的数组，都被映射为 Hprose 列表类型。例如：

```
$array = array(1, 2, 3, 4, 5);
```

数组是否映射为列表类型只与索引有关，与元素值无关，所以元素值可以是同一种类型，也可以是不同类型。

字典类型

所有除映射为 Hprose 列表类型以外的数组类型均映射为 Hprose 字典类型，例如：

```
$a = array( 1 => 'one', 2 => 'two', 3 => 'three' );
$map = array( 'version'    => 4,
              'OS'        => 'Linux',
              'lang'       => 'english',
              'short_tags' => true
            );
```

都被映射为 Hprose 字典类型。

另外 PHP 的 stdClass 对象也被映射为字典类型，例如从数据库中查询出的每一行数据。

对象类型

PHP 中自定义类的对象实例在序列化时被映射为 Hprose 对象类型。自定义类中的字段名，映射为 Hprose 对象类型中的属性名，自定义类中的字段值，映射为 Hprose 对象类型中的属性值。所有的字段必须为可序列化类型，在 PHP 中除了资源类型以外的其它类型均为可序列化类型。

PHP 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 PHP 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 HproseClassManager 来注册自定义类型

在 Hprose 1.2 for PHP 中，通过 HproseClassManager 的 register 方法可以让你不改变类名定义就可以与其它语言进行交互。

例如您有一个命名为 User 的类，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');
```

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 PHP 服务器，在本章中您将深入的了解 Hprose for PHP 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，除了提供了跟 PHPRPC 中相同的 add 方法以外，还提供了更多方便的方法。

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，甚至包括 PHP 中的内置的函数。但如果参数或结果中包含有资源类型（比如 mysql_connect，mysql_query 等），那么这种函数就不能够发布。

你可以同时发布多个函数，不论是你自定义的，还是 PHP 内置的都可以。例如：

```
<?php
include("hprose/hproseHttpServer.php");
$server = new HproseHttpServer();
$server->addFunction("trim");
$server->addFunctions(array('md5', 'sha1'));
$server->handle();
?>
```

在上面这个例子中我们发布了三个方法，这三个都是 PHP 内置的函数。我们可以把方法名放到数组中，用 addFunctions 一次添加多个，也可以以字符串作为 addFunction 方法的参数一个一个的添加。

发布方法

Hprose for PHP 也支持发布类的静态方法和对象的实例方法，如下例：

```
<?php
include("hprose/hproseHttpServer.php");
class Example1 {
    static function foo() {
        return 'foo';
    }
    function bar() {
        return 'bar';
    }
}
$server = new HproseHttpServer();
$server->addMethod('foo', 'Example1');
$server->addMethod('bar', new Example1());
$server->handle();
?>
```

这里 foo 是一个静态方法，所以添加时第二个参数是类名。而 bar 是一个实例方法，所以添加时是一个 Example1 的实例对象。

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：

```
<?php
include("hprose/hproseHttpServer.php");
function hello($name) {
    return 'Hello ' . $name;
}
class Example1 {
    static function foo() {
        return 'foo';
    }
    function bar() {
        return 'bar';
    }
}
class Example2 {
    function foo() {
        return 'foo, too';
    }
    function bar() {
        return 'bar, too';
    }
}
$server = new HproseHttpServer();
$server->addFunction('hello', 'hi');
$server->addMethod('foo', 'Example1', 'ex1_foo');
$server->addMethod('bar', new Example1(), 'ex1_bar');
$server->addMethods(array('foo', 'bar'), new Example2(), array('ex2_foo', 'ex2_bar'));
$server->handle();
?>
```

从上面这个例子，我们就会发现不论是函数还是方法都可以通过别名来发布，只需要在最后再加一个别名参数就可以了。同时添加多个方法（或函数）时，别名也应该是多个，并且个数要跟方法（或函数）名个数相同，且一一对应。

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。

发布对象

除了向上面通过 addMethod 和 addMethods 发布方法以外，Hprose 可以让您更方便的发布一个对象上的方法，那就是使用 addInstanceMethods，addInstanceMethods 用来发布指定对象上的指定类层次上声明的所有 public 实例方法。它有三个参数，其中后两个是可选参数。如果您在使用 addInstanceMethods 方法时，不指定类层次（或者指定为 NULL），则发布这个对象所在类上声明的所有 public 实例方法。这个方法也支持指定名称空间（别名前缀）。

例如：

```
<?php
include("hprose/hproseHttpServer.php");
class Example2 {
    function foo() {
        return 'foo, too';
    }
    function bar() {
        return 'bar, too';
    }
}
$server = new HproseHttpServer();
$server->addInstanceMethods(new Example2(), NULL, 'ex2');
$server->handle();
?>
```

效果是跟上面别名机制例子中发布 ex2_foo 和 ex2_bar 方法一样的。

发布类

跟 addInstanceMethods 方法类似，使用 addClassMethods 可以让您更方便的发布一个类上的方法。它有三个参数，其中后两个是可选参数。第一个参数是方法的发布类，第二个参数为方法的执行类，第三个参数为名称空间（别名前缀）。

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 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 发布的方法可以是实例方法、静态方法或者函数，但是只能发布一个。如果多次调用 addMissingFunction 方法，将只有最后一次发布的有效。

用 addMissingFunction 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 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。

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 PHP 客户端，在本章中您将深入的了解 Hprose for PHP 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 参数设置
- 保持会话

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：

```
<?php
include("hprose/hproseHttpClient.php");
$client = new HproseHttpClient("http://www.hprose.com/example/");
echo "<pre>";
echo $client->sum(1, 2, 3, 4, 5);
echo "\r\n";
echo $client->Sum(6.0, 7.0, 8.0);
echo "\r\n";
$userList = $client->getUserList();
print_r($userList);
echo "</pre>";
?>
```

这个例子的运行结果是：

```
15
21
Array
(
    [0] => User Object
        (
            [name] => Amy
            [sex] => 2
            [birthday] => HproseDate Object
                (
                    [year] => 1983
                    [month] => 12
                    [day] => 3
                    [utc] =>
                )

            [age] => 26
            [married] => 1
        )

    [1] => User Object
        (
            [name] => Bob
            [sex] => 1
            [birthday] => HproseDate Object
```

```
            (
                [year] => 1989
                [month] => 6
                [day] => 12
                [utc] =>
            )

        [age] => 20
        [married] =>
    )

[2] => User Object
    (
        [name] => Chris
        [sex] => 0
        [birthday] => HproseDate Object
            (
                [year] => 1980
                [month] => 3
                [day] => 8
                [utc] =>
            )

        [age] => 29
        [married] => 1
    )

[3] => User Object
    (
        [name] => Alex
        [sex] => 3
        [birthday] => HproseDate Object
            (
                [year] => 1992
                [month] => 6
                [day] => 14
                [utc] =>
            )

        [age] => 27
        [married] =>
    )

)
```

从这个例子中，我们可以看出通过远程方法名进行调用时，远程方法名是不区分大小写的，所以不论是写 sum 还是 Sum 都可以正确调用，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 会自动帮你生成这个类的定义(例如上例中的 User 类),并返回这个类的对象。

通过 Invoke 方法进行远程调用

非引用参数传递

上面介绍的通过远程方法名进行远程调用的例子就是非引用参数传递，下面是用 invoke 方法重写的代码：

```
<?php
include("hprose/hproseHttpClient.php");
$client = new HproseHttpClient("http://www.hprose.com/example/");
echo "<pre>";
$args = array(1, 2, 3, 4, 5);
echo $client->invoke("sum", $args);
echo "\r\n";
$args = array(6.0, 7.0, 8.0);
echo $client->invoke("Sum", $args);
echo "\r\n";
$userList = $client->invoke("getUserList");
print_r($userList);
echo "</pre>";
?>
```

运行结果与上面例子的运行结果完全相同。但是我们发现用 invoke 方法并不方便，因为当有参数时，必须要把参数单独放入一个变量中才可以进行传递。所以通常我们无需直接使用 invoke 方法，除非我们需要动态调用。另外，还有一种情况下，你会用到 invoke 方法，那就是在进行引用参数传递时。

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：

```
<?php
include("hprose/hproseHttpClient.php");
$client = new HproseHttpClient("http://www.hprose.com/example/");
echo "<pre>";
$args = array(array("Mon"=>1,"Tue"=>2,"Wed"=>3,"Thu"=>4,"Fri"=>5,"Sat"=>6,"Sun"=>7));
echo "args(before invoke):\r\n";
print_r($args);
$result = $client->invoke("swapKeyAndValue", $args, true);
echo "args(after invoke):\r\n";
print_r($args);
```

```
echo "result:\r\n";
print_r($result);
echo "</pre>";
?>
```

上面程序的运行结果如下：

```
args(before invoke):
Array
(
    [0] => Array
        (
            [Mon] => 1
            [Tue] => 2
            [Wed] => 3
            [Thu] => 4
            [Fri] => 5
            [Sat] => 6
            [Sun] => 7
        )

)
args(after invoke):
Array
(
    [0] => Array
        (
            [1] => Mon
            [2] => Tue
            [3] => Wed
            [4] => Thu
            [5] => Fri
            [6] => Sat
            [7] => Sun
        )

)
result:
Array
(
    [1] => Mon
    [2] => Tue
    [3] => Wed
    [4] => Thu
    [5] => Fri
```

```
    [6] => Sat
    [7] => Sun
)
```

我们看到运行前后，$args 中的值已经改变了。

这里有一点要注意，当参数本身是数组时，该数组应该作为参数数组的第一个元素传递，例如上例中的$args 是：

```
array(array("Mon"=>1,"Tue"=>2,"Wed"=>3,"Thu"=>4,"Fri"=>5,"Sat"=>6,"Sun"=>7))
```

而不能只写：

```
array("Mon"=>1,"Tue"=>2,"Wed"=>3,"Thu"=>4,"Fri"=>5,"Sat"=>6,"Sun"=>7)
```

否则程序将会出错，或者在调用中陷入等待状态，这样的错误不容易被找到，因此一定要注意这一点。

异常处理

Hprose for PHP 的客户端只支持同步调用，因此在调用过程中，如果服务器端发生错误，异常将在客户端被直接抛出，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。

例如，当调用不存在的方法时：

```
<?php
include("hprose/hproseHttpClient.php");
$client = new HproseHttpClient("http://www.hprose.com/example/");
echo "<pre>";
try {
  echo $client->unexistMethod();
}
catch (Exception $e) {
  print_r($e);
}
echo "</pre>";
?>
```

运行结果如下：

```
HproseException Object
(
    [message:protected] => Can't find this function unexistmethod().
file: D:\phpox\hprose\www\example\hproseHttpServer.php
line: 152
trace: #0 D:\phpox\hprose\www\example\hproseHttpServer.php(374): HproseHttpServer->doInvok
```

```
e()
#1 D:\phpox\hprose\www\example\hproseHttpServer.php(391): HproseHttpServer->handle()
#2 D:\phpox\hprose\www\example\index.php(58): HproseHttpServer->start()
#3 {main}
    [string:private] =>
    [code:protected] => 0
    [file:protected] => /var/www/hprose/hproseHttpClient.php
    [line:protected] => 165
    [trace:private] => Array
        (
            [0] => Array
                (
                    [file] => /var/www/hprose/hproseHttpClient.php
                    [line] => 191
                    [function] => invoke
                    [class] => HproseHttpClient
                    [type] => ->
                    [args] => Array
                        (
                            [0] => unexistMethod
                            [1] => Array
                                (
                                )

                        )

                )

            [1] => Array
                (
                    [function] => __call
                    [class] => HproseHttpClient
                    [type] => ->
                    [args] => Array
                        (
                            [0] => unexistMethod
                            [1] => Array
                                (
                                )

                        )

                )

            [2] => Array
```

```
            (
                [file] => /var/www/exam4.php
                [line] => 6
                [function] => unexistMethod
                [class] => HproseHttpClient
                [type] => ->
                [args] => Array
                    (
                    )

            )

        )

)
```

超时设置

Hprose 1.2 for PHP 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 setTimeout 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。

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 来设置 http 代理服务器的地址和端口。参数是字符串，例如："tcp://10.54.1.39:8000"，默认值为 NULL。

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，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 setHeader 方法来设置 Cookie 标头，将会影响 Session 的正常工作。

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。通过 setKeepAlive 设置为 true 时，表示开启持久连接特征。setKeepAliveTimeout 可以设置持久连接超时时间，单位是秒，默认值是 300 秒。

保持会话

当浏览器访问一个 PHP 页面，而这个 PHP 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 PHP 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：

```
session_start();
include("hproseHttpClient.php");
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 PHP 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。