Submitted by **gouki** on 2008, August 2, 8:03 PM. PHP

PHP 序列化（serialize）格式详解

1．前言

PHP （从 PHP 3.05 开始）为保存对象提供了一组序列化和反序列化的函数：serialize、unserialize。不过在 PHP 手册中对这两个函数的说明仅限于如何使用，而对序列化结果的格式却没做任何说明。因此，这对在其他语言中实现 PHP 方式的序列化来说，就比较麻烦了。虽然以前也搜集了一些其他语言实现的 PHP 序列化的程序， 不过这些实现都不完全，当序列化或反序列化一些比较复杂的对象时，就会出错了。于是我决定写一份关于 PHP 序列化格式详解的文档（也就是这一篇文档），以便在编写其他语言实现的 php 序列化程序时能有一个比较完整的参考。这篇文章中所写的内容是我通过编写程序测试和阅读 PHP 源代码得到的，所以，我不能 100% 保证所有的内容都是正确的，不过我会尽量保证我所写下的内容的正确性，对于我还不太清楚的地方，我会在文中明确指出，也希望大家能够给予补充和完善。

2．概述

PHP 序列化后的内容是简单的文本格式，但是对字母大小写和空白（空格、回车、换行等）敏感，而且字符串是按照字节（或者说是 8 位的字符）计算的，因此，更合适的说法是 PHP 序列化后的内容是字节流格式。因此用其他语言实现时，如果所实现的语言中的字符串不是字节储存格式，而是 Unicode 储存格式的话，序列化后的内容不适合保存为字符串，而应保存为字节流对象或者字节数组，否则在与 PHP 进行数据交换时会产生错误。

PHP 对不同类型的数据用不同的字母进行标示，Yahoo 开发网站提供的 Using Serialized PHP with Yahoo! Web Services 一文中给出所有的字母标示及其含义：

- a - array
- b - boolean
- d - double
- i - integer
- o - common object
- r - reference
- s - string
- C - custom object
- O - class
- N - null
- R - pointer reference
- U - unicode string

N 表示的是 NULL，而 b、d、i、s 表示的是四种标量类型，目前其它语言所实现的 PHP 序列化程序基本上都实现了对这些类型的序列化和反序列化，不过有一些实现中对 s （字符串）的实现存在问题。

« 2019年04月 »

日	一	二	三	四	五	六
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30				

日志分类

PHP [608]
python [9]
Go [39]
Scala [9]
lua [2]
Javascript [248]
PHP Framework [63]
Linux [239]
苹果相关 [185]
DataBase [157]
Software [231]
Literature [8]
Ideas [17]
產品 [7]
Misc [901]
Baby [92]

热门标签

mysql php jquery yii ubuntu google linux javascript firefox 肖佑阳 framework database phpstorm thinkphp svn mac apache zend typecho wordpress qq 连载 seo chrome 优化 ipad netbeans ios css 架构

日志归档

2019年04月 [2]
2019年03月 [8]
2019年02月 [4]
2019年01月 [8]
2018年12月 [3]

更多...

搜索文章

确定

高级搜索

最新评论

IE不是全面淘汰了吗？像我这么支持微软的人，也早就不用IE了。现...
04-08 - watermoon

请问你怎么解决的。----我有提到啊。需要在routers里定义
03-21 - 66

交换链接吗
https://www.yd631.com/
02-16 - 美国主机

单纯路过
02-10 - xuxu

frp官方的issue里有解决方案，就是需要nginx本地用个r...
01-28 - nickfan

更多...

博客信息

分类数量: 16
文章数量: 2814
评论数量: 1871
标签数量: 2215
附件数量: 939
注册用户: 56
今日访问: 6574
总访问量: 25962181
程序版本: 1.6

友情链接

囧囧孙
CoolCode
PHPText
蜜糖公社
板子博客
有意思NIS
快递查询
iOrange
简单人生
google上的neatstudio

更多...

a、O 属于最常用的复合类型，大部分其他语言的实现都很好的实现了对 a 的序列化和反序列化，但对 O 只实现了 PHP4 中对象序列化格式，而没有提供对 PHP 5 中扩展的对象序列化格式的支持。

r、R 分别表示对象引用和指针引用，这两个也比较有用，在序列化比较复杂的数组和对象时就会产生带有这两个标示的数据，后面我们将详细讲解这两个标示，目前这两个标示尚没有发现有其他语言的实现。

C 是 PHP5 中引入的，它表示自定义的对象序列化方式，尽管这对于其它语言来说是没有必要实现的，因为很少会用到它，但是后面还是会对它进行详细讲解的。

U 是 PHP6 中才引入的，它表示 Unicode 编码的字符串。因为 PHP6 中提供了 Unicode 方式保存字符串的能力，因此它提供了这种序列化字符串的格式，不过这个类型 PHP5、PHP4 都不支持，而这两个版本目前是主流，因此在其它语言实现该类型时，不推荐用它来进行序列化，不过可以实现它的反序列化过程。在后面我也会对它的格式进行说 明。

最后还有一个 o，这也是我唯一还没弄清楚的一个数据类型标示。这个标示在 PHP3 中被引入用来序列化对象，但是到了 PHP4 以后就被 O 取代了。在 PHP3 的源代码中可以看到对 o 的序列化和反序列化与数组 a 基本上是一样的。但是在 PHP4、PHP5 和 PHP6 的源代码中序列化部分里都找不到它的影子，但是在这几个版本的反序列化程序源代码中却都有对它的处理，不过把它处理成什么我还没弄清楚。因此对它暂时不再 作更多说明了。

补充

最近的 PHP CVS 版本中，序列化的方式有所变化，基本类型的序列化仍然保持原来的格式，只是对 Unicode 支持上，有了新的进展。另外，对普通字符串的序列化也分成了 2 种。一种是 non-escaped 字符串，也就是我们上面说的那个小写 s 标识的字符串；另一种是 escaped 字符串，这种字符串格式用大写 S 标识。所以上面那个表现在应该改为：

- a - array
- b - boolean
- d - double
- i - integer
- o - common object
- r - reference
- s - non-escaped binary string
- S - escaped binary string
- C - custom object
- O - class
- N - null
- R - pointer reference
- U - unicode string

3．NULL 和标量类型的序列化

NULL 和标量类型的序列化是最简单的，也是构成符合类型序列化的基础。这部分内容相信许多 PHP 开发者都已经熟知。如果您感觉已经掌握了这部分内容，可以直接跳过这一章。

3.1．NULL 的序列化

在 PHP 中，NULL 被序列化为：

N;

3.2．boolean 型数据的序列化

boolean 型数据被序列化为：

b:<digit>;

其中 <digit> 为 0 或 1，当 boolean 型数据为 false 时，<digit> 为 0，否则为 1。

3.3．integer 型数据的序列化

integer 型数据（整数）被序列化为：

i:<number>;

其中 <number> 为一个整型数，范围为：-2147483648 到 2147483647。数字前可以有正负号，如果被序列化的数字超过这个范围，则会被序列化为浮点数类型而不是整型。如果序列化后的数字超过这个范围（PHP 本身序列化时不会发生这个问题），则反序列化时，将不会返回期望的数值。

3.4．double 型数据的序列化

double 型数据（浮点数）被序列化为：

d:<number>;

其中 <number> 为一个浮点数，其范围与 PHP 中浮点数的范围一样。可以表示成整数形式、浮点数形式和科学技术法形式。如果序列化无穷大数，则 <number> 为 INF，如果序列化负无穷大，则 <number> 为 -INF。序列化后的数字范围超过 PHP 能表示的最大值，则反序列化时返回无穷大（INF），如果序列化后的数字范围超过 PHP 所能表示的最小精度，则反序列化时返回 0。当浮点数为非数时，被序列化为 NAN，NAN 反序列化时返回 0。但其它语言可以将 NAN 反序列化为相应语言所支持的 NaN 表示。

3.5．string 型数据的序列化

string 型数据（字符串）被序列化为：

s:<length>:"<value>";

其中 <length> 是 <value> 的长度，<length> 是非负整数，数字前可以带有正号（+）。<value> 为字符串值，这里的每个字符都是单字节字符，其范围与 ASCII 码的 0 - 255 的字符相对应。每个字符都表示原字符含义，没有转义字符，<value> 两边的引号（""）是必须的，但不计算在 <length> 当中。这里的 <value> 相当于一个字节流，而 <length> 是这个字节流的字节个数。

补充

在 PHP5 最新的 CVS 中（也就是将来要发布的 PHP6），上面对于 string 型数据的序列化方式已经被下面这种所取代，但是 PHP6 仍然支持上面那种序列化方式的反序列化。

新的序列化方式叫做 escaped binary string 方式，这是相对与上面那种 non-escaped binary string 方式来说的：

string 型数据（字符串）新的序列化格式为：

S:<length>:"<value>";

其中 <length> 是源字符串的长度，而非 <value> 的长度。<length> 是非负整数，数字前可以带有正号（+）。<value> 为经过转义之后的字符串。

它的转义编码很简单，对于 ASCII 码小于 128 的字符（但不包括 \），按照单个字节写入（与 s 标识的相同），对于 128~255 的字符和 \ 字符，则将其 ASCII 码值转化为 16 进制编码的字符串，以 \ 作为开头，后面两个字节分别是这个字符的 16 进制编码，顺序按照由高位到低位排列，也就是第 8-5 位所对应的16进制数字字符（abcdef 这几个字母是小写）作为第一个字节，第 4-1 位作为第二个字节。依次编码下来，得到的就是 <value> 的内容了。

个人认为这种大 S 方式的序列化没有小 s 方式好，对于反序列化来说，小 s 方式可以获得更快的速度。所以，我认为 PHP6 的这种修改是 PHP6 的一大败笔！如果要在其它语言实现 PHP 序列化，只要支持小 s 方式序列化就可以了，使用大 S 方式序列化，将与目前主流的 PHP4 和 PHP5 不兼容。但是反序列化应该实现大 S 方式的反序列化，这样可以兼容 PHP6 序列化的字符串内容。

4．简单复合类型的序列化

PHP 中的复合类型有数组（array）和对象（object）两种，本章主要介绍在简单情况下这两种类型数据的序列化格式。关于嵌套定义的复合类型和自定义序列化方式的对象的序列化格式将在后面的章节详细讨论。

4.1．数组的序列化

数组（array）通常被序列化为：

a:<n>:{<key 1><value 1><key 2><value 2>...<key n><value n>}

其中 <n> 表示数组元素的个数，<key 1>、<key 2>……<key n> 表示数组下标，<value 1>、<value 2>……<value n> 表示与下标相对应的数组元素的值。

下标的类型只能是整型或者字符串型（包括后面那种 Unicode 字符串型），序列化后的格式跟整型和字符串型数据序列化后的格式相同。

数组元素值可以是任意类型，其序列化后的格式与其所对应的类型序列化后的格式相同。

4.2．对象的序列化

对象（object）通常被序列化为：

O:<length>:"<class name>":<n>:{<field name 1><field value 1><field name 2><field value 2>...<field name n><field value n>}

其中 <length> 表示对象的类名 <class name> 的字符串长度。<n> 表示对象中的字段[1]个数。这些字段包括在对象所在类及其祖先类中用 var、public、protected 和 private 声明的字段，但是不包括 static 和 const 声明的静态字段。也就是说只有实例（instance）字段。

<filed name 1>、<filed name 2>……<filed name n>表示每个字段的字段名，而 <filed value 1>、<filed value 2>……<filed value n> 则表示与字段名所对应的字段值。

字段名是字符串型，序列化后格式与字符串型数据序列化后的格式相同。

字段值可以是任意类型，其序列化后的格式与其所对应的类型序列化后的格式相同。

但字段名的序列化与它们声明的可见性是有关的，下面重点讨论一下关于字段名的序列化。

4.3．对象字段名的序列化

var 和 public 声明的字段都是公共字段，因此它们的字段名的序列化格式是相同的。公共字段的字段名按照声明时的字段名进行序列化，但序列化后的字段名中不包括声明时的变量前缀符号 $。

protected 声明的字段为保护字段，在所声明的类和该类的子类中可见，但在该类的对象实例中不可见。因此保护字段的字段名在序列化时，字段名前面会加上

\0*\0

的前缀。这里的 \0 表示 ASCII 码为 0 的字符，而不是 \0 组合。

private 声明的字段为私有字段，只在所声明的类中可见，在该类的子类和该类的对象实例中均不可见。因此私有字段的字段名在序列化时，字段名前面会加上

\0<declared class name>\0

的前缀。这里 <declared class name> 表示的是声明该私有字段的类的类名，而不是被序列化的对象的类名。因为声明该私有字段的类不一定是被序列化的对象的类，而有可能是它的祖先类。

字段名被作为字符串序列化时，字符串值中包括根据其可见性所加的前缀。字符串长度也包括所加前缀的长度。其中 \0 字符也是计算长度的。

---

[1]注：在 PHP 手册中，字段被称为属性，而实际上，在 PHP 5 中引入的用 __set、__get 来定义的对象成员更适合叫做属性。因为用 __set、__get 来定义的对象成员与其它语言中的属性的行为是一致，而 PHP 手册中所说的属性实际上在其他语言中（例如：C#）中被称为字段，为了避免混淆，这里也称为字段，而不是属性。

5．嵌套复合类型的序列化

上一章讨论了简单的复合类型的序列化，大家会发现对于简单的数组和对象其实也很容易。但是如果遇到自己包含自己或者 A 包含 B，B 又包含 A 这类的对象或数组时，PHP 又该如何序列化这种对象和数组呢？本章我们就来讨论这种情况下的序列化形式。

5.1．对象引用和指针引用

在 PHP 中，标量类型数据是值传递的，而复合类型数据（对象和数组）是引用传递的。但是复合类型数据的引用传递和用 & 符号明确指定的引用传递是有区别的，前者的引用传递是对象引用，而后者是指针引用。

在解释对象引用和指针引用之前，先让我们看几个例子。

```
<?php
echo "<pre>";
class SampleClass {
   var $value;
}
$a = new SampleClass();
$a->value = $a;

$b = new SampleClass();
$b->value = &$b;

echo serialize($a);
echo "\n";
echo serialize($b);
echo "\n";
echo "</pre>";
?>
```

这个例子的输出结果是这样的：

```
O:11:"SampleClass":1:{s:5:"value";r:1;}
O:11:"SampleClass":1:{s:5:"value";R:1;}
```

大家会发现，这里变量 $a 的 value 字段的值被序列化成了 r:1，而 $b 的 value 字段的值被序列化成了 R:1。

但是对象引用和指针引用到底有什么区别呢?

大家可以看下面这个例子：

```
echo "<pre>";
class SampleClass {
   var $value;
}
$a = new SampleClass();
$a->value = $a;

$b = new SampleClass();
$b->value = &$b;

$a->value = 1;
$b->value = 1;

var_dump($a);
var_dump($b);
echo "</pre>";
```

大家会发现，运行结果也许出乎你的预料：

```
object(SampleClass)#1 (1) {
  ["value"]=>
  int(1)
}
int(1)
```

改变 $a->value 的值仅仅是改变了 $a->value 的值，而改变 $b->value 的值却改变了 $b 本身，这就是对象引用和指针引用的区别。

不过很不幸的是，PHP 对数组的序列化犯了一个错误，虽然数组本身在传递时也是对象引用传递，但是在序列化时，PHP 似乎忘记了这一点，看下面的例子：

```
echo "<pre>";
$a = array();
$a[1] = 1;
$a["value"] = $a;

echo $a["value"]["value"][1];
echo "\n";
$a = unserialize(serialize($a));
echo $a["value"]["value"][1];
echo "</pre>";
```

结果是：

```
1
```

大家会发现，将原数组序列化再反序列化后，数组结构变了。原本 $a["value"]["value"][1] 中的值 1，在反序列化之后丢失了。

原因是什么呢？让我们输出序列化之后的结果来看一看：

```
$a = array();
$a[1] = 1;
$a["value"] = $a;

echo serialize($a);
```

结果是：

```
a:2:{i:1;i:1;s:5:"value";a:2:{i:1;i:1;s:5:"value";N;}}
```

原来，序列化之后，$a["value"]["value"] 变成了 NULL，而不是一个对象引用。

也就是说，PHP 只对对象在序列化时才会生成对象引用标示（r）。对所有的标量类型和数组（也包括 NULL）序列化时都不会生成对象引用。但是如果明确使用了 & 符号作的引用，在序列化时，会被序列化为指针引用标示（R）。

5.2．引用标示后的数字

在上面的例子中大家可能已经看到了，对象引用（r）和指针引用（R）的格式为：

```
r:<number>;
R:<number>;
```

大家一定很奇怪后面这个 <number> 是什么吧？本节我们就来详细讨论这个问题。

这个 <number> 简单的说，就是所引用的对象在序列化串中第一次出现的位置，但是这个位置不是指字符的位置，而是指对象（这里的对象是泛指所有类型的量，而不仅限于对象类型）的位置。

我想大家可能还不是很明白，那么我来举例说明一下：

```
class ClassA {
    var $int;
    var $str;
    var $bool;
    var $obj;
    var $pr;
}

$a = new ClassA();
$a->int = 1;
$a->str = "Hello";
$a->bool = false;
$a->obj = $a;
$a->pr = &$a->str;
```

```
echo serialize($a);
```

这个例子的结果是：

```
O:6:"ClassA":5:{s:3:"int";i:1;s:3:"str";s:5:"Hello";s:4:"bool";b:0;s:3:"obj";r:1;s:2:"pr";R:3;}
```

在这个例子中，首先序列化的对象是 ClassA 的一个对象，那么给它编号为 1，接下来要序列化的是这个对象的几个成员，第一个被序列化的成员是 int 字段，那它的编号就为 2，接下来被序列化的成员是 str，那它的编号就是 3，依此类推，到了 obj 成员时，它发现该成员已经被序列化了，并且编号为 1，因此它被序列化时，就被序列化成了 r:1; ，在接下来被序列化的是 pr 成员，它发现该成员实际上是指向 str 成员的一个引用，而 str 成员的编号为 3，因此，pr 就被序列化为 R:3; 了。

PHP 是如何来编号被序列化的对象的呢？实际上，PHP 在序列化时，首先建立一个空表，然后每个被序列化的对象在被序列化之前，都需要先计算该对象的 Hash 值，然后判断该 Hash 值是否已经出现在该表中了，如果没有出现，就把该 Hash 值添加到这个表的最后，返回添加成功。如果出现了，则返回添加失败，但是在返回失败前先判断该对象是否是一个引用（用 & 符号定义的引用），如果不是则也把 Hash 值添加到表后（尽管返回的是添加失败）。如果返回失败，则同时返回上一次出现的位置。

在添加 Hash 值到表中之后，如果添加失败，则判断添加的是一个引用还是一个对象，如果是引用，则返回 R 标示，如果是对象，则返回 r 标示。因为失败时，会同时返回上一次出现的位置，因此，R 和 r 标示后面的数字，就是这个位置。

5.3．对象引用的反序列化

PHP 在反序列化处理对象引用时很有意思，如果反序列化的字符串不是 PHP 的 serialize() 本身生成的，而是人为构造或者用其它语言生成的，即使对象引用指向的不是一个对象，它也能正确地按照对象引用所指向的数据进行反序列化。例如：

```
echo "<pre>";
class StrClass {
   var $a;
   var $b;
}

$a = unserialize('O:8:"StrClass":2:{s:1:"a";s:5:"Hello";s:1:"b";r:2;}');

var_dump($a);
echo "</pre>";
```

运行结果：

```
object(StrClass)#1 (2) {
  ["a"]=>
  string(5) "Hello"
  ["b"]=>
  string(5) "Hello"
}
```

大家会发现，上面的例子反序列化后，$a->b 的值与 $a->a 的值是一样的，尽管 $a->a 不是一个对象，而是一个字符串。因此如果大家用其它语言来实现序列化的话，不一定非要把 string 作为标量类型来处理，即使按照对象引用来序列化拥有相同字符串内容的复合类型，用 PHP 同样可以正确的反序列化。这样可以更节省序列化后的内容所占用的空间。

6．自定义对象序列化

6.1．PHP 4 中自定义对象序列化

PHP 4 中提供了 __sleep 和 __wakeup 这两个方法来自定义对象的序列化。不过这两个函数并不改变对象序列化的格式，影响的仅仅是被序列化字段的个数。关于它们的介绍，在 PHP 手册中写的还算比较详细。这里就不再多做介绍了。

6.2．PHP 5 中自定义对象序列化

PHP 5 中增加了接口（interface）功能。PHP 5 本身提供了一个 Serializable 接口，如果用户在自己定义的类中实现了这个接口，那么在该类的对象序列化时，就会被按照用户实现的方式去进行序列化，并且序列化后的标示不再是 O，而改为 C。C 标示的格式如下：

```
C:<name length>:"<class name>":<data length>:{<data>}
```

其中 <name length> 表示类名 <class name> 的长度，<data length> 表示自定义序列化数据 <data> 的长度，而自定义的序列化数据 <data> 是完全的用户自己定义的格式，与 PHP 序列化格式可以完全无关，这部分数据由用户自己实现的序列化和反序列化接口方法来管理。

Serializable 接口中定义了 2 个方法，serialize() 和 unserialize($data)，这两个方法不会被直接调用，而是在调用 PHP 序列化函数时，被自动调用。其中 serialize 函数没有参数，它的返回值就是 <data> 的内容。而 unserialize($data) 有一个参数 $data，这个参数的值就是 <data> 的内容。这样大家应该就明白了，实际上接口中 serialize 方法就是让用户来自己序列化对象中的内容，序列化后的内容格式，PHP 并不关心，PHP 只负责把它充填到 <data> 中，等到反序列化时，PHP 只负责取出这部分内容，然后传给用户实现的 unserialize($data) 接口方法，让用户自己去反序列化这部分内容。

下面举个简单的例子，来说明 Serializable 接口的使用：

```
class MyClass implements Serializable
{
    public $member;

    function MyClass()
    {
        $this->member = 'member value';
    }

    public function serialize()
    {
        return wddx_serialize_value($this->member);
    }

    public function unserialize($data)
    {
        $this->member = wddx_deserialize($data);
    }
}
$a = new MyClass();
echo serialize($a);
echo "\n";
print_r(unserialize(serialize($a)));
```

输出结果为（浏览器中的源代码）：

```
C:7:"MyClass":90:{<wddxPacket version='1.0'><header/><data><string>member value</string></data>
</wddxPacket>}
MyClass Object
(
    [member] => member value
)
```

因此如果想用其它语言来实现 PHP 序列化中的 C 标示的话，也需要提供一种这样的机制，让用户自定义类时,能够自己在反序列化时处理 <data> 内容，否则，这些内容就无法被反序列化了。

7．Unicode 字符串的序列化

好了，最后再谈谈 PHP 6 中关于 Unicode 字符串序列化的问题吧。

说实话，我不怎么喜欢把字符串搞成双字节 Unicode 这种编码的东西。JavaScript 中也是用这样的字符串，因此在处理字节流的东西时，反而非常的不方便。C# 虽然也是用这种方式来编码字符串，不过还好的是，它提供了全面的编码转换机制，而且提供这种字符串到字节流（实际上是到字节数组）的转换，所以处理起来还 算是可以。但是对于不熟悉这个的人来说，转来转去就是个麻烦。

PHP 6 之前一直是按字节来编码字符串的，到了 PHP 6 突然冒出个 Unicode 编码的字符串来，虽然是可选的，但仍然让人觉得非常不舒服，如果配置不当，老的程序兼容性都成问题。

当然加了这个东西以后，许多老的与字符串有关的函数都进行了修改。序列化函数也不例外。因此，PHP 6 中增加了专门的 Unicode 字符串序列化标示 U。PHP 6 中对 Unicode 字符串的序列化格式如下：

U:<length>:"<unicode string>";

这里 <length> 是指原 Unicode String 的长度，而不是 <unicode string> 的长度，因为 <unicode string> 是经过编码以后的字节流了。

但是还有一点要注意，<length> 尽管是原 Unicode String 的长度，但是也不是只它的字节数，当然也不完全是指它的字符数，确切的说是之它的字符单位数。因为 Unicode String 中采用的是 UTF16 编码，这种编码方式使用 16 位来表示一个字符的，但是并不是所有的都是可以用 16 位表示的，因此有些字符需要两个 16 位来表示一个字符。因此，在 UTF16 编码中，16 位字符算作一个字符单位，一个实际的字符可能就是一个字符单位，也有可能由两个字符单位组成。因此， Unicode String 中字符数并不总是等于字符单位数，而这里的 <length> 指的就是字符单位数，而不是字符数。

那 <unicode string> 又是怎样被编码的呢？实际上，它的编码也很简单，对于编码小于 128 的字符（但不包括 \），按照单个字节写入，对于大于 128 的字符和 \ 字符，则转化为 16 进制编码的字符串，以 \ 作为开头，后面四个字节分别是这个字符单位的 16 进制编码，顺序按照由高位到低位排列，也就是第 16-13 位所对应的16进制数字字符（abcdef 这几个字母是小写）作为第一个字节，第 12-9 位作为第二个字节，第 8-5 位作为第三个字节，最后的第 4-1 位作为第四个字节。依次编码下来，得到的就是 <uncode string> 的内容了。

我认为对于其他语言来说，没有必要实现这种序列化方式，因为用这种方式序列化的内容，对于目前的主流 PHP 服务器来说都是不支持的，不过倒是可以实现它的反序列化，这样将来即使跟 PHP 6 进行数据交换，也可以互相读懂了。

8．参考文献

PHP 3 中关于序列化和反序列化的源代码

PHP 4 中关于序列化的源代码

PHP 4 中关于反序列化的源代码

PHP 5 中关于序列化的源代码

PHP 5 中关于反序列化的源代码

PHP 6 中关于序列化的源代码

PHP 6 中关于反序列化的源代码

PHP 手册中关于序列化和反序列化的介绍

Using Serialized PHP with Yahoo! Web Services

本站采用创作共享版权协议, 要求署名、非商业和保持一致. 本站欢迎任何非商业应用的转载, 但须注明出自"臆叔", 保留原始链接, 此外还必须标注原文标题和链接.

Tags: php tutorials, serialize

« 上一篇 | 下一篇 »

相关文章 只显示10条记录

关于序列化javascript (浏览: 12361, 评论: 0)
hprose使用中的一個問題 (浏览: 12023, 评论: 1)
yii CActiveRecord 中的一点小注意事项 (浏览: 11084, 评论: 1)

访客评论 1条记录

iamheretotellyouastorythatiloveu

Post by soitrievn on 2008, August 6, 5:51 AM #1

发表评论

名字 (必填):

密码 (游客不需要密码):

网址或电子邮件 (选填):

评论内容 (必填):

提交

友情链接 | 合作站点 | 其他网站 |
360网站安全检测平台
https://neatstudio.com ::: Copyright © 2004-2006 膘叔
Processed in 0.695396 second(s), 8 queries, Gzip disabled XHTML 1.0. 清除Cookies. 又有谁的博客已经永久消失了?