Conceptos Básicos php III. Cadenas

Heike Bonilla Redondo

Índice

1. Conceptos básicos

Lo único que puede un ordenador son bits de información, cuyos valores son 0 y 1. Para que los bits representen algo se utilizan reglas para convertir una secuencia de bits en algo con significado como letras o números, para ello utiliza un esquema de codificación. Por ejemplo, en ASCII la letra c es 01100011, la letra s es 01110011. Una secuencia de ocho bits forma un byte, y representa una letra. En ASCII existen 128 caracteres, ya que utiliza 7 bits para la codificación y el octavo para paridad. Entre los caracteres que permite representar se encuentran las letras de la A a la Z en mayúsculas y en minúsculas, los números del 1 al 9, puntuaciones, símbolos (dólar $, ampersand &...), espacio, tabulador... 128 es el número máximo de combinaciones diferentes de ceros y unos que se pueden obtener con 7 bits (0000001, 0000011, 0101110...).

En la tabla de representación de caracteres de ASCII se puede observar que 128 bits no son suficientes para trabajar con diferentes idiomas: alemán, ruso, chino mandarín, japonés... Como ASCII sólo emplea 7 bits y en cada ordenador se trabaja con 8 bits por byte, si se utiliza ese bit restante es posible ampliar a 255 combinaciones diferentes, lo que aumenta el rango. Así se hizo, y la versión extendida de ASCII incluye caracteres especiales del latín. Pero aun así 255 caracteres no son suficientes ni para cubrir las variaciones de letras y puntuaciones de los idiomas de los países europeos.

2. Codificaciones multibyte

Las necesidades expuestas en el apartado anterior hace que aparezcan las codificaciones multibyte.

Empezaron a surgir múltiples sistemas de codificación diferentes para cubrir esta necesidad pero diferentes métodos suponían un caos para la comunicación entre sistemas, entonces surge la necesidad de un estándar que los unifique, así es como apareció Unicode, permite representar cualquier tipo de símbolo o letra de cualquier idioma.

Normalmente existen diferentes formas de codificar Unicode: UTF-32 utiliza todos los puntos posibles de Unicode usando 32 bits, es decir, 4 bytes por carácter. Esto emplea mucho espacio en muchos casos. UTF-16 y UTF-8 son codificaciones de amplitud variable. Esto significa que si un carácter se puede representar con un sólo byte, UTF-8 empleará sólo un byte. Si requiere dos bytes, usará dos, y así sucesivamente.

Carácter	Codificación	Bits
A	UTF-8	01000001
A	UTF-16	00000000 01000001
A	UTF-32	00000000 00000000 00000000 01000001

Formato de codificación	Descripción
UTF-8	Sistema de codificación de 8 bits, compatible con ASCII, amplitud variable
UTF-16	Sistema de codificación de 16 bits, amplitud variable
UTF-32	Sistema de codificación de 32 bits, amplitud fija

Cuando se trabaja con **caracteres multibyte** se suele trabajar con **UTF-8**, que además de ser **compatible con ASCII**, codifica caracteres en **múltiples bytes** con **amplitud variable**, es decir, que si puede emplear un byte para un carácter empleará un byte. El **UTF-32** por ejemplo emplea 4 bytes para cualquier carácter.

3. Interpretación de la codificación

Cuando abres un documento de texto en un editor o una página web en el navegador y aparecen caracteres como â€™ y otros caracteres extraños es que hay un error en la interpretación de la codificación.

El problema es básicamente que el editor o navegador que está intentando leer el documento está utilizando una codificación diferente a la que se ha utilizado para crearlo. Para solucionarlo, simplemente se selecciona el esquema de codificación correcto.

Lo más importante de UTF-8 es que es compatible binariamente con ASCII. Todos los caracteres disponibles en ASCII sólo utilizan un byte en UTF-8 y son los mismos bits. Cualquier otro carácter que no está en ASCII, emplea dos o más bytes en UTF-8. En la mayoría de lenguajes de programación que esperan interpretar ASCII se puede incluir texto en UTF-8 directamente en los programas.

4. Codificación en PHP

PHP no soporta de forma nativa a Unicode, pero UTF-8 es totalmente compatible con ASCII, que es todo lo que PHP necesita.

Cuando un lenguaje de programación no soporta nativamente Unicode, se refiere a que el lenguaje asume que un carácter equivale a un byte. Por ejemplo, PHP permite acceso directo a los caracteres de un string mediante la notación de arrays:

```
echo $cadena[0];
```

Si $cadena está codificada con codificación monobyte, nos da el primer carácter, que equivale a un byte. PHP nos devuelve el primer byte sin pensar acerca de caracteres. Los strings son secuencias de bytes para PHP, nada más.

El hecho de que PHP no soporte de forma nativa a Unicode simplemente significa que la mayoría de funciones PHP interpretan un byte por carácter, lo que produce que los caracteres multibyte se corten. Esto no significa que no se pueda usar Unicode en PHP o que todo string en Unicode tenga que ser traducido con _utf8_encode_.

Las funciones utf8_encode y utf8_decode_ son inútiles si no se quiere convertir específicamente de ISO-8859-1 a UTF-8 y viceversa (si se quiere convertir entre cualquier otro tipo de esquema, hay que utilizar iconv).

Es por ello que existe la extensión **mbstring**, o Multibyte String extension, que hace una réplica de casi todas las funciones de strings importantes de PHP en modo multibyte. En cada función de **mbstring**, como **mb_substr**, se puede especificar el esquema de codificación:

```
mb_substr($string, 0, 1, 'UTF-8');
```

5. Comparación de strings con operadores

A la hora de **comparar variables**, resulta sencillo cuando se trata de ***booleanos*** o de ***integers***, pero cuando se trata de ***strings*** o partes de *strings* se puede complicar un poco.

La forma más común de comparar dos strings es con el **operador ==**. Si los dos strings son iguales, el condicional devuelve true:

```
if('hola' == 'hola'){
   echo "El string coincide";
} else {
   echo "El string no coincide";
}
```

Este tipo de comparación es sensible a mayúsculas y minúsculas, por lo que *'hola' == 'Hola'* devolvería **false**.

Si todavía se quiere ser más estricto, se pueden emplear tres signos de igual, **===**. De esta forma el tipo de **valor** del que se trata tiene que ser igual:

```
if( 4 === '4'){
   echo "El string coincide";
} else {
   echo "El string no coincide";
}
```

En este caso la condición resulta **false**, ya que son dos tipos distintos, un *integer* y un *string*. Deberían ser '4' === '4' o 4 === 4.

5.1.- Algunas funciones que facilitan la comparación de caracteres.

strcmp

```
int strcmp (string $str1, string $str2)
```

Es la función más básica para comparar dos strings a nivel binario. Tiene en cuenta mayúsculas y minúsculas. Devuelve < 0 si el primer valor dado es menor que el segundo, > 0 si es al revés, y 0 si son iguales:

```
$var1 = "hola";
$var2 = "hola";
if (strcmp($var1, $var2) ==0){
   echo "Los strings coinciden";
}
```

En la práctica sólo se suelen utilizar estas funciones utilizando estructuras de control condicionales respecto al valor 0, como en el ejemplo anterior y en los siguientes.

strcasecmp

```
int strcasecmp (string $str1, string str2)
```

Idéntica a la anterior pero esta vez insensible a mayúsculas y minúsculas. Devuelve los mismos resultados:

```
$var1 = "Hola";
$var2 = "hola";
if (strcmp($var1, $var2) === 0){
 echo "Los strings coinciden";
                                                    }
```

strncmp

```
int strncmp (string $str1, string $str2, int $length)
```

Comparación a nivel binario de los primeros **n** caracteres entre strings. Es igual que *strcmp()*, solo que se puede especificar un **límite de caracteres** de cada string a comparar. Es sensible a mayúsculas y minúsculas, y devuelve el mismo tipo de resultado:

```
$var1 = "holG34d";
$var2 = "holaquetal";
if (strncmp($var1, $var2, 3) === 0){
            echo "Los strings coinciden";
} // Coinciden, ya que hol === hol
```

strncasecmp

```
int strncasecmp (string $str1, string $str2, int $length)
```

Idéntico a *strncmp()* pero insensible a mayúsculas y minúsculas. Devuelve lo mismo que las anteriores:

```
$var1 = "hoLAQG34d";
$var2 = "holaQuetal";
if (strncasecmp($var1, $var2, 5) === 0){
   echo "Los strings coinciden";
} // Los strings también coinciden
```

6. Acceso y modificación de strings por caracteres

Se puede acceder y modificar los **caracteres de un string** especificando el índice de base cero del carácter deseado, como si fuera un *array*:

```
$cadena = "animal";
echo $cadena[1]; // Devuelve: n
```

```
// Pueden emplearse llaves también:
echo $cadena{3}; // Devuelve: m
```

Un string es como un array de caracteres. Internamente **los strings son arrays de bytes**, por lo que acceder o modificar un string mediante corchetes o llaves no es seguro con **caracteres multibyte** que sean diferentes de los que incluye ASCII.

Si el **valor del índice** es negativo o mayor que el propio string, se emite un error E_NOTICE. Sólo se puede devolver el primer carácter señalado.

```
$cadena = "Blog sobre programación";
$quinto = $cadena[5]; // Devuelve: s
$cuarto = $cadena[4]; // Devuelve: espacio en blanco
```

El problema surge cuando queremos mostrar por ejemplo una vocal con tilde, en este caso no funcionará este método.

7. Funciones incorporadas para la extracción de strings

Existen diversas funciones incorporadas en **PHP** para extraer porciones de strings de otros strings, extraer strings de strings para formar arrays, extraer los espacios de un string, etc. Se dividen en tres secciones principales: extracción parcial de strings, extracciones relacionadas con arrays y extracciones de elementos.

Extracción parcial

Funciones que extraen partes de un string.

substr

```
string substr (string $str, int $start [, int $length])
```

Devuelve parte de un string definido por los parámetros *$start* y *$length*. El parámetro *$start* indica la posición del carácter desde la cual se comenzará la extracción, por ejemplo en 'perro', el carácter en 0 es p, en 1 es e, etc. Si *$start* es **negativo** se comienza desde el final del string.

Si la **longitud del string** es menor que *$start*, la función devuelve **false**.

```
$str = substr("perro", -1); // Devuelve o
$str = substr("perro", -4); // Devuelve erro
$str = substr("calabacín", 3, 2); // Devuelve ab
$str = substr("pupitre", -4, 3); // Devuelve itr
$str = substr("gato", 5, 2); // Devuelve false
```

Si se especifica *$length*, el string devuelto contendrá tantos caracteres como el número indicado. Si *$length* es mayor que el número de caracteres existente en el string desde donde extraer, se extraerá sólo el máximo existente.

Si *$length* es **negativo**, se omite ese mismo número de caracteres del final del string devuelto.

Si no se especifica *$length*, el string devuelto será desde *$start* hasta el final.

Otros ejemplos de uso:

```
$str = substr("diego", 0, -1); // Devuelve: dieg
$str = substr("cascada", 5, -2); // Devuelve: string vacío
$str = substr("cascada", 5, -3); // Devuelve false
```

No acepta caracteres multibyte:

$str = substr("almacén", -2, 2); // Devuelve: ?n

Para que se pueda extraer un string con **caracteres multibyte** se puede usar la función _mb*substr()*.

mb_substr

```
string mb_substr (string $str, int $start [, int $length = NULL [, string $encoding = mb_internal_encoding() ]])
```

Obtiene parte de una cadena de caracteres. Es igual que *substr()* pero se puede especificar la **codificación de caracteres**. Si se omite, se usará el valor de la **codificación de caracteres interna**.

explode

```
array explode (string $delimeter, string $string [, int $limit])
```

Divide un string *$string* y lo convierte en un array. Cada uno de los strings formados por la división *$delimeter* son elementos del array que se crea.

El límite *$limit* indica el número máximo de elementos que devolverá, y el último elemento contendrá el resto del string. Si limit es **negativo**, se devolverán todos los elementos a excepción de los *-$limit* últimos. Si *$limit* es 0 se tratará como 1.

Devuelve el *array* de *strings* creados por la división *$delimeter*. Si *$delimeter* está vacío, *explode()* devuelve **false**.

Si no se encuentra el delimitador en el string, se devuelte un array con un único elemento, el cual será el mismo string.

```
$pastel = "trozo1, trozo2 con mermelada, trozo3 con naranja, trozo4";
$trozos = explode(", ", $pastel);
echo $trozos[1]; // Devuelve trozo2 con mermelada
```

implode

```
string implode (array $pieces)
```

Une los elementos de un array *$pieces* en un string con el pegamento *$glue* que se indique.

Devuelve un string con todos los elementos del array en el mismo orden, separados por el pegamento.

Si se usa un **array vacío** devuelve un **string vacío**.

```
$animales = array('perro', 'gato', 'cerdo', 'vaca', 'gallina', 'escorpión', 'mono');
echo $conjunto_animales = implode(",", $animales);
```

str_split

```
array str_split (string $string [, int $split_length = 1])
```

Convierte un string *$string* en un array.

_$split*length* es opcional. Si se especifica, el *array* devuelto se separará en fragmentos que tendrán una longitud igual a su valor. Si no se especifica, el array será sólo de un elemento, cuyo valor será el de *$string*. Si es menor que uno, devolverá **false**.

Si trata con **strings con caracteres multibyte**, realizará la **división en bytes** en lugar de en caracteres, lo que suele dar errores:

Comprobar que ocurre con caracteres en UTF-8 direfentes a los ASCII

trim

```
string trim (string $str [, string $character_mask = " \t\n\r\0\x0B" ])
```

Elimina espacios en blanco u otros caracteres **al principio y al final de la cadena**. Si no se especifica el segundo parámetro, se tienen en cuenta todos los siguientes: **\t\n\r\0\x0B** (espacio, tabulación, salto de línea, retorno de carro, el byte null y tabulación vertical). Si se especifica sólo se retirarán los indicados.

```
$cadena1 = "\tEste texto es de prueba\n";
$cadena2 = " Este también";

$trimmed1 = trim($cadena1);
$trimmed2 = trim($cadena2);

echo $trimmed1;
echo $trimmed2;
```

ltrim

```
string ltrim (string $str [, string $character_mask])
```

Función igual que la anterior, pero sólo retira los espacios y otros caracteres del **principio del string**.

```
$cadena1 = "\tEste texto es de prueba\t";
$trimmed = ltrim($cadena1, "\t");
```

// En el resultado sólo se ha retirado \t del principio, no del final:

rtrim

```
string rtrim (string $str [, string $character_mask ])
```

Función igual que trim, pero sólo retira los espacios y otros caracteres del **final del string**.

```
$cadena1 = "\tEste texto es de prueba\t";
$trimmed = rtrim($cadena1, "\t");
```

// En el resultado sólo se ha retirado \t del final, no del principio:

8. Equivalencias de mbstring con funciones de strings

Para **funciones básicas de manipulación de strings**, como *strlen()* o *substr()*, existen versiones compatibles con **caracteres multibyte**. Si no se usan estas versiones surgen resultados inesperados y errores de codificación.

Tabla de equivalencias de funciones de strings y mbstring:

Monobyte	Multibyte	Descripción
strlen()	mb_strlen	Devuelve la longitud del string
strpos()	mb_strpos()	Encuentra la posición de la primera ocurrencia de un string en otro
substr()	mb_substr()	Devuelve parte de un string
strtolower()	mb_strtolower()	Devuelve un string en minúsculas
strtoupper()	mb_strtoupper()	Devuelve un string en mayúsculas
substr_count()	mb_substr_count()	Cuenta el número de ocurrencias de un substring
split()	mb_split()	Divide un string en un array mediante una expresión regular

Las funciones preg_* vienen con compatibilidad con UTF-8 por defecto.

Existen más **funciones compatibles**, las puedes ver en esta lista de funciones para strings multibyte.