

## 第 3 章 字符串和数组操作

第 2 章初步介绍了 PHP 的数据类型，本章将详细地介绍其中的字符串和数组的操作。

### 3.1 操作字符串

在 Web 应用中，用户和系统的交互基本上是用文字来进行的，因此系统对文本信息，即字符串的处理非常重要。文本字符串操作内容很多，本节将对其一一介绍。

#### 3.1.1 去除空格和其他特殊符号

有时，需要去掉字符串中的空格或者其他没有意义的符号。例如，在一个电子商务应用中，当用户填写订单的内容时（如联系地址），可能输入一些空格、句号等字符，系统希望在存储之前把它们去掉，只剩下干净有意义的信息。为了完成类似于上述需求的问题，PHP4 及以上版本提供了 4 个去除字符串中特殊符号的函数：

`string trim ( string str [, string charlist])`：去除字符串 `str` 首尾处空格或其他特殊符号；

`string ltrim ( string str [, string charlist])`：去除字符串 `str` 首的空格或其他特殊符号；

`string rtrim ( string str [, string charlist])`：去除字符串 `str` 尾的空格或其他特殊符号；

`string chop ( string str [, string charlist])`：同 `rtrim()`。

以上函数的第一个参数 `str` 为待操作的字符串，第二个可选参数 `charlist` 指定想要去除的特殊符号，当缺省时默认值为去掉下列字符：空格（" "）、制表符（\t）、换行符（\n）、回车符（\r）、空值（\0）。还可以使用“..”在通过第二个参数指定需要去除一个范围内的字符，例如“a..d”指去掉 ASCII 码值介于 a 和 d 之间的字符，即 a、b、c、d。

下面以 `trim()` 为例说明上述函数的使用：

```
1  <!--去除字符串中的特殊字符：trim.php-->
2  <?php
3      $init_str = ",山东省济南市经十路 8 号 1."; //前后都包含一个空格
4      echo $init_str."#<br>";
5      $trimmed_str = trim($init_str);           //"山东省济南市经十路 8 号 1."
6      echo $trimmed_str."#<br>";
7      $trimmed_str = trim($init_str, ',.');//"山东省济南市经十路 8 号 1"。注意，第二个参数包括
三个字符。
8      echo $trimmed_str."#<br>";
9      $trimmed_str = trim($init_str, '0..9');//"山东省济南市经十路 8 号"。0..9 说明要去掉所有的
数字字符
10     echo $trimmed_str."#<br>";
11  ?>
```

第 3 行定义了一个字符串变量，在其首部有空格和逗号，其尾有句号和空格；

第 5 行使用不带第二个参数的 `trim()` 函数去掉了其中首尾处的空格符号；

第 7 行使用带有第二个参数的 `trim()` 函数去掉了其首尾处的第二个参数中所包含的字符，即去掉了首尾处的空格、逗号和句号。

第 9 行 trim() 中第二个参数中的 “ 0..9 ” 说明将要去掉这个位于 0 和 9 的 ASC 码范围内的所有字符。

ltrim()、 rtrim() 的使用同 trim() 类似，而 chop() 实际上是 rtrim() 的别名，其功能与 rtrim() 一样，不再赘述。

### 3.1.2 加入和去除反斜杠

在许多应用中，例如生成 SQL 语句时（SQL 语句将在第二部分中介绍），需要在其中加入转义字符 ‘ \ ’，手工构造起来相当麻烦。为了解决类似问题，PHP 提供了自动在字符串中加入或去除转义字符的函数：

string addslashes ( string str, string charlist)：第一个参数 str 为待操作的原始字符串，第二个参数 charlist 说明需要在原始串的那一些字符前加上字符 ‘ \ ’。

string stripslashes ( string str)：去掉字符串中的 ‘ \ ’。

二者的使用参考下面代码：

```
1 <!--加入转义字符：addslashes.php-->
2 <?php
3 $init_str = "select * from Books where name = 'PHP 手册'";
4 echo $init_str."#<br>";
5 $new_str = addslashes($init_str,"");
6 echo $new_str."#<br>";
7 $init_str2 = stripslashes($new_str);
8 echo $init_str2."#<br>";
9 ?>
```

代码在第 5 行在 \$init\_str 中的 ‘ ’ 前加上了 ‘ \ ’，又在第 9 行将其去掉。

### 3.1.3 生成 HTML 元素

HTML 元素的书写非常麻烦，下面简单列出一些常用字符在 HTML 中的表示方式。

'&'：'&amp;';

双引号 ‘ ” ’：'&quot;';

单引号 ‘ ’ ’：'&#039;';

'<'：'&lt;';

'>'：'&gt;';

此处，称‘&’等为 HTML 元素，‘&amp;’等为其显示字符串。例如，若想在页面上的显示 “ <a href='test'>链接</a> ”，HTML 应写为“ &lt;a href=&#039;test&#039;&gt;Test&lt;/a&gt; ”，否则，将只在页面上显示一个链接信息。

PHP 提供了下面的函数来自动转化 HTML 元素：

string htmlspecialchars ( string str [, int quote\_style [, string charset]])：把一些常用的 HTML 元素转换为显示字符串；

string htmlentities ( string str [, int quote\_style [, string charset]])：把所有的 HTML 元素转换为显示字符串；

string html\_entity\_decode ( string str [, int quote\_style [, string charset]])：把显示字符串转化为 HTML 元素。

上面函数中，参数 str 表示原始字符串；可选参数 quote\_style 确定是否转换双引号和单引号，取值范围为{ ENT\_COMPAT, ENT\_QUOTES, ENT\_NOQUOTES}，分别表示只转换双引号、

全转换、全不转换，缺省时默认值为 ENT\_COMPAT；第三个参数 charset 指定了转换中所用的字符集。PHP4 及以上版本所支持的字符集参考表 3.1。

表 3.1 PHP4 及以上版本支持的字符集

字符集	说明
ISO-8859-1	西欧字符集
ISO-8859-15	西欧字符集扩展
UTF-8	兼容ASCII的宽字节字符集
cp1252	西欧字符集，Windows系统默认
BIG5	繁体中文，用于中国台湾省
GB2312	简体中文，用于中国大陆
BIG5-HKSCS	繁体中文扩展，用于中国香港
Shift_JIS	日文
EUCJP	日文

下面的示例中，首先使用 htmlentities()函数得到一个 HTML 语句的显示字符串，然后再用 html\_entity\_decode()函数重新把显示字符串转回 HTML 元素。运行结果如图 3.1 所示。

```
1 <!--生成 HTML 元素：htmlspecialchars.php-->
2 <?php
3 $orig = "我正在<b><font color=red>学习！</font></b> ";
4 $a = htmlentities($orig,ENT_COMPAT,"GB2312");
5 $b = html_entity_decode($a);
6 echo $a; // I'll &quot;walk&quot; the &lt;b&gt;dog&lt;/b&gt; now
7 echo $b; // I'll "walk" the <b>dog</b> now
8 ?>
```



图 3.1 PHP 生成 HTML 元素示例

注意：函数 html\_entity\_decode()只支持 PHP4.0.3 及以上版本。  
除上面所提到的三个函数之外，用于 HTML 元素操作的函数还包括 nl2br()、get\_html\_translation\_table()等，功能与上述函数类似，本书不再一一详述。

3.1.4 分解字符串

分解字符串是指把一个字符串通过特殊的符号分解为许多子串。例如，时间字符串“2005-01-01 12:59:59”可以利用符号“-”、空格和“:”分解为年月日时分秒具体的值。PHP 提供了下列函数完成类似功能：

```
array split ( string pattern, string str [, int limit] );
```

其中,参数 pattern 指定了作为分解标识的符号 ;str 为待操作的原始串 ;第三个可选参数 limit 为返回子串个数的最大值，缺省时为全部返回。函数的返回值为数组，将在 3.2 节对其进行介绍。此处，可以暂时把函数返回值理解为多个子串即可。

下面的示例可以把字符串“2005-01-01 12:59:59”分解为年月日时分秒子串：

```
1 <!--分解字符串：split.php-->
2 <?php
3 $date = "2005-01-01 12:59:59";
4 list ($year,$month,$day,$hour,$minute,$second) = split ('[- :]', $date);
5 echo "{$year}年{$month}月{$day}日{$hour}时{$minute}分{$second}秒<br>\n";
```

```
6    ?>
```

上例将输出“2005年01月01日12时59分59秒”。第4行使用split函数把时间分解，分解的标识符包括“-”、空格和“:”，在第5行将其输出。

除split之外，功能相似的函数还包括preg\_split()、explode()、implode()、chunk\_split()和wordwrap()等。

### 3.1.5 格式化字符串

格式化字符串用于按一定的格式输出含有许多变量的文本，是最常用的一种操作。PHP的sprintf()函数完成这个功能，习惯使用C语言的读者肯定对其感到熟悉。函数原型为：

```
string sprintf(string format, mixed [args]...);
```

参数format是转换后的格式，各个变量都以“%”后的字符规定其格式，后面的多个参数以此对应于format中的“%”处。下面示例格式化浮点数的小数部分。

```
1    <!--格式化字符串：sprintf.php-->
2    <?
3    $name="张三";
4    $money1 = 68.75;
5    $money2 = 54.35;
6    $money = $money1 + $money2;
7    // 此时变数 $money 值为 "123.1";
8    $formatted = sprintf ("%s 有 ￥ %01.2f。", $name, $money);
9    echo $formatted;    //张三有 ￥ 123.10。
10   ?>
```

第6行通过算术运算，得到\$money的值为123.1；而在第8行通过sprintf中的%01.2定义其格式为显示小数点后两位。

除sprintf()之外，常用于格式化数据的函数还有printf()、sprintf()、sscanf()、fscanf()、vsprintf()、number\_format()等。

### 3.1.6 获取和替换子串

获取子串是指对从一个串中获取其中连续的一部分。例如，从串“2005-01-01 12:59:59”中取得时间串。PHP提供了两个函数来获取或替换串的某一部分：

string substr ( string str, int start [, int length]) ：获取子串，第一个参数str是待操作的串，第二个参数start表明子串在总串中的起始位置，第三个可选参数指定所获取的子串长度，如果为正数则表明子串从start向右取，否则向左取；缺省时默认值为从start取到串尾。

string substr\_replace ( string str, string replacement, int start [, int length]) :在获取的基础上进行替换，即将获取出的子串替换为其第二个参数replacement。

下面的示例中，首先利用substr()获取串“2005-01-01 12:59:59”的时间信息，然后使用substr\_replace()函数将年份信息改为“2006”：

```
1    <!--获取子串：substr.php-->
2    <?
3    $date = "2005-01-01 12:59:59";
4    $time=substr($date,11,8); //子串"12:59:59"的起始位置为 11，长度为 8
5    echo "time:$time<br>";
6    $new_date=substr_replace($date,"2006",0,4);
7    echo "new date:$new_date";
8    ?>
```

### 3.1.7 定位字符

定位字符是指寻找某个字符在串中最先出现的位置，函数 `strpos()` 可以完成此功能：

`int strpos ( string str, char needle)`：第一个参数 `str` 为待处理的串，第二个参数 `needle` 为待寻找的字符。下面这个示例，对一个电子邮件地址进行处理，首先使用 `strpos()` 寻找字符 “@”，然后结合获取子串函数 `substr()` 获取用户名。

```
1 <!--字符定位：strpos.php-->
2 <?
3     $email = "zhangsan@php.net";
4     $i=strpos($email,'@');
5     $name=substr($email,0,$i);
6     echo $name;
7 ?>
```

示例第 4 行使用 `strpos()` 获取了字符 ‘@’ 的位置，然后在第 5 行使用 `substr()` 得到用户名子串信息。

### 3.1.8 求串长度

求串长度也是常用的操作，所使用的函数为 `strlen()`：`int strlen ( string str)`。

这个函数很简单，返回字符串 `str` 的长度。仍以上一小节的例子为例，从电子邮件串中替换用户的名字，即改为 “lisi@php.net”：

```
1 <!--字符定位：strpos.php-->
2 <?
3     $email = "zhangsan@php.net";
4     $i=strpos($email,'@');
5     $name=substr($email,0,$i);
6     $email=substr_replace($email,"lisi",0,strlen($name));
7     echo $email;
8 ?>
```

### 3.1.9 获取 ASCII 编码

把字符转化为 ASCII 编码在实际应用中有时是很有用的，例如，字符串在数据库中以二进制形式存放，而需要数据获取函数返回 ASCII 码串时，就需要把其转化为字符串显示。PHP 提供的转换 ASCII 码和字符的函数有：

`string chr ( int ascii)`：把 ASCII 码转化为字符串；

`int ord ( string string)`：把字符串转化为 ASCII 码。

二者的使用参考下例：

```
1 <!--ASCII 转换：chr.php-->
2 <?
3     $letter = chr(65);    //A
4     $ascii=ord('A');    //65
5     echo $letter;
6     echo $ascii;
7 ?>
```

### 3.1.10 比较字符串

字符串的比较规则是按照字典排序方法，排在前面的小于后面的。如同在一本英语词典中，后面的词条大于前面的词条。PHP 实现字符串比较的函数为：

`int strncmp ( string str1, string str2[, int len]`)：函数的前两个参数为待比较的两个字符串，第三个可选参数可指定想比较二者从头开始的多少个字符。如果 `str1>str2`，函数返回正数；`str1=str2` 时返回 0；`str1<str2` 时返回负数。

参考下例：

```
1 <!--ASCII 转换：chr.php-->
2 <?
3     $str1="China";
4     $str2="Beijing";
5     $i=strcmp($str1,$str2);
6     echo $i;        //1
7.  ?>
```

除 `strcmp()` 之外，具有字符串比较或排序功能的函数还 `strcasecmp()`、`strncmp()`、`strncasecmp()`、`strnatcasecmp()`、`strstr()`、`natsort()` 和 `natcasesort()`。

### 3.1.11 大小写转换

当比较两个字符串是否在不区分大小写时相等时，仅仅使用上一小节的 `strcmp()` 函数就不行了，这时可将两个字符串同时转换为大写或小写，然后再进行比较即可。例如，在判断网站登录的用户名和密码（不区分大小写时）时，常需要这样。PHP 实现字符串大小写转换的函数有：

`string strtolower ( string str )`：将 `str` 转换为小写形式；  
`string strtoupper ( string string )`：将 `str` 转换为大写形式；  
`string ucfirst ( string str )`：将 `str` 的第一个字符转换为大写形式；  
`string ucwords ( string str )`：将 `str` 中每一个单词的首字母转换为大写形式。

参考下例：

```
1 <!--大小写转换：Upper_Lower.php-->
2 <?
3     $str1="shandong province";
4     $str2="China";
5     $str1=ucwords($str1);
6     echo $str1;        //Shangdong Province
7     $str1=strtoupper($str1);
8     echo $str1;        //SHANGDONG PROVINCE
9     $str2=strtolower($str2);
10    echo $str2;        //china
11  ?>
```

### 3.1.12 小结

字符串是 PHP 中应用最为广泛的数据类型，其操作也种类繁多。PHP4 及以上版本提供了五十多个内置的字符串操作函数，熟练的使用这些函数，是使用 PHP 的重要内容。本节介绍了其中最为常用的大部分函数，其余未能涉及的部分，请读者在开发或学习过程中参考 PHP 函数手册。

## 3.2 正则表达式

正则表达式是一个非常大的题目 PHP 继承了 Perl 的正则表达式法则 ,还有自己的一套法则。本节将详细介绍 PHP 的正则表达式。

### 3.2.1 理解正则表达式

正则表达式是一种可以用于模式匹配的强大工具。简单地说,正则表达式就是一套规则,用于去判定其他的元素是否符合它。

举一个简单的例子:在一个用户注册的页面中(例如,一个论坛或者交友网站的注册页面),上面可能有“电子邮件”这一项需要填写。对系统来说,需要判定用户所填写的电子邮件地址是否合法,即是否符合电子邮件地址的规则。利用上一节介绍的字符串操作技术可以来实现这个功能:

```
1  <!--检查电子邮件合法性: validate_email1.php-->
2  <?php
3  function validate_email1($email)
4  {
5      $hasAtSymbol = strpos($email, "@"); //检查是否包含@
6      $hasDot = strpos($email, "."); //检查是否包含.
7      if($hasAtSymbol && $hasDot && $hasAtSymbol<$hasDot )
8          return 1;
9      else
10         return 0;
11 }
12 echo validate_email1("tom@php.net"); //true
13 echo validate_email1("tom@php"); //false
14 ?>
```

上面代码实现了一个函数 `validate_email1()`, 使用字符串操作中的定位字符函数, 用来判断一个字符串是否是一个合法的电子邮件地址。仔细考虑实现的功能, 实际上是在判断一个字符串是否具有一定的模式, 或者说是否满足一定的规则。在这种情况下, 就可以使用正则表达式来实现相同的功能:

```
1  <!--使用正则表达式检查电子邮件合法性: validate_email2.php-->
2  <?php
3  function validate_email2($email)
4  {
5      return (int)ereg("^([a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+)$", $email);
6  }
7  echo validate_email2("tom@php.net"); //true
8  echo validate_email2("tom@php"); //false
9  ?>
```

上面实现了具有相同功能的函数 `validate_email1()`, 函数使用了一个正则表达式的函数 `ereg()`, 其具体使用将在下一节介绍。

观察 `ereg()` 函数的参数 “`^([a-zA-Z]+@[a-zA-Z]+\.[a-zA-Z]+)$`”, 容易看出其实际上表示满足这样规则的字符串: 以`[a-zA-Z]`即任意大小写字母字符串开头, 然后紧跟 “`@`”, 然后又是任意大小写字母组成的字符串, 第四部分是符号 “`.`”, 最后仍是任意字符串。这相当于定义了一个字符串的组成规则。

在看过这个示例后，重新来看正则表达式的定义：正则表达式是一种可以用于模式匹配的强大工具。

### 3.2.2 使用正则表达式

在 PHP 有六个函数来处理正则表达式，用来检查一个字符串是否满足一个的规则。它们都把一个正则表达式作为它们的第一个参数，语法为：

`bool ereg ( string pattern, string string [, array regs])`：最常用的正则表达式函数，搜索跟正则表达式 `pattern` 匹配的一个字符串。搜索到返回 `true`，否则返回 `false`。

`string ereg_replace ( string pattern, string replacement, string string)`：搜索跟正则表达式 `pattern` 匹配的一个字符串，并用新的字符串代替所有这个表达式出现的地方。

`bool eregi ( string pattern, string string [, array regs])`：和 `ereg` 几乎是一样效果，不过忽略大小写。

`string eregi_replace ( string pattern, string replacement, string string)`：和 `ereg_replace` 有着一样的搜索-替换功能，不过忽略大小写。

`array split ( string pattern, string string [, int limit])`：搜索和正则表达式匹配的字符串，并且以字符串集合的方式返回匹配结果。

`array spliti ( string pattern, string string [, int limit])`：同 `split` 一样，不过忽略大小写。

本节，旨在给出 PHP 提供的正则表达式函数，并简单介绍其功能。对于上述函数的具体使用，则要在首先了解了正则表达式的构造之后才能介绍。

### 3.2.3 构造正则表达式

如前所述，在匹配一个字符串到正则表达式之前，必须先构造正则表达式。

#### 1. 定义头部规则

PHP 用 “^” 定义字符串头部的规则，例如：“^hello” 即定义头部为 “hello” 的字符串，结合上一节所介绍的函数，代码

```
<?php echo ereg("^hello", "hello world!"); ?> //true
```

将返回 “true”，因为待验证的字符串 “hello world!” 满足规则：以 “hello” 开头。而

```
<?php echo ereg("^hello", "i say hello world!"); ?> //false
```

将返回 `false`，因为 `hello` 不在字符串 “I say hello world” 的头部。

#### 2. 定义尾部规则

PHP 用 “\$” 定义字符串尾的规则，例如：“world\$” 即定义尾部为 “world” 的字符串，代码

```
<?php echo ereg("world$", "hello world!"); ?> //true
```

将返回 “true”，因为待验证的字符串 “hello world!” 满足规则：以 “world” 结尾。而

```
<?php echo ereg("world$", "i say hello php"); ?> //false
```

将返回 `false`，因为 `world` 不在字符串 “I say hello php” 的尾部。

#### 3. 定义包含任意字符规则

PHP 用 “.” 定义包含任意字符的规则，例如：“.” 即定义包含任意字符的字符串，代码

```
<?php echo ereg(".", "something"); ?> //true
```

将返回 “true”，因为待验证的字符串 “something” 满足规则：包含任意字符。而



```
<?php echo ereg(".", ""); ?> //false
```

将返回 false，因为空串不包含任意字符。

#### 4. 定义包含字符数目规则

使用大括号 “{n}” 定义包含 n 个任意字符，“{m,n}” 定义包含 m 到 n 范围内的任意字符。例如 “. {3}” 定义包含 3 个连续字符的字符串，“. {1,3}” 定义包含 1 到 3 个字符的字符串。代码

```
<?php echo ereg(".{3}", "abcd"); ?> //true
```

```
<?php echo ereg(".{1,3}", "aa"); ?> //true
```

都将返回 “true”，因为待验证的字符串 “aaa” 满足规则：包含三个字符。而

```
<?php echo ereg(".{4}", "aaa"); ?> //false
```

都将返回 false。

“.” 也可以结合 “^” 和 “\$” 使用，例如 “. {3}\$” 定义了以三个字符结尾的字符串，“^.{3}” 定义了以三个连续字符开头的字符串。代码

```
<?php echo ereg(".{3}$", "baaa"); ?> //true
```

将返回 “true”，因为待验证的字符串 “baaa” 满足规则：以三个连续字符结尾。而

```
<?php echo ereg(".{3}$", "ab"); ?> //false
```

都将返回 false。

#### 5. 定义包含 0-n 个字符规则

PHP 用 “\*” 定义包含 0-n 个字符的规则，例如：“a\*” 即定义包含 0-n 个字符 “a” 的字符串，代码

```
<?php echo ereg("a*", "aaa"); ?> //true
```

将返回 “true”，因为待验证的字符串 “aaa” 满足规则：包含 0-n 个字符 “a”。而

```
<?php echo ereg("a*", "bbb"); ?> //true
```

也将返回 true，因为待验证的字符串 “bbb” 也满足规则：包含 0-n 个字符 “a”。

#### 6. 定义包含 1-n 个字符规则

PHP 用 “+” 定义包含 1-n 个字符的规则，例如：“a+” 即定义包含 1-n 个字符 “a” 的字符串，代码

```
<?php echo ereg("a+", "aaa"); ?> //true
```

将返回 “true”，因为待验证的字符串 “aaa” 满足规则：包含 1-n 个字符 “a”。而

```
<?php echo ereg("a+", "bbb"); ?> //false
```

将返回 false，因为待验证的字符串 “bbb” 不满足规则：包含 1-n 个字符 “a”。

#### 7. 定义包含 0 或 1 个字符规则

PHP 用 “?” 定义包含 0 或 1 个字符的规则，例如：“a?” 即定义包含 0 或 1 个字符 “a” 的字符串，代码

```
<?php echo ereg("a?", "a"); ?> //true
```

```
<?php echo ereg("a?", "bb"); ?> //true
```

将返回 “true”，因为待验证的字符串 “a” 和 “bb” 都满足规则：包含 0 或 1 个字符 “a”。而

```
<?php echo ereg("a?", "aa"); ?> //false
```

将返回 false，因为待验证的字符串 “bbb” 不满足规则：包含 0 或 1 个字符 “a”。

#### 8. 定义包含某范围的字符规则

PHP 用方括号 “[start-end]” 定义包含 start-end 范围内任意字符的规则，例如：“[a-z]” 即定义包含 a 到 z 范围内任意字符的字符串，代码

```
<?php echo ereg("[a-z]+$", "abc"); ?> //true
```

将返回 “true”，因为待验证的字符串 “abc” 满足规则：包含 1-n 个字符 a 到 z 范围内任意

字符。

```
<?php echo ereg("[a-z]+$", "ABC"); ?> //false
```

将返回 false，因为待验证的字符串“bbb”不满足规则：包含 1-n 个字符“a”。

### 9. 定义包含某范围的词规则

PHP 用圆括号“(word1|word2|...)”定义包含 word1、word2、... 的任意字符串的规则，例如：

“(wang|zhang)”即定义包含“wang”或“zhang”的任意字符的字符串，代码

```
<?php echo ereg("^ (wang|zhang) +$", "wang and zhang"); ?> //true
```

将返回“true”，因为待验证的字符串“wang and zhang”满足规则：以“wang”或“zhang”开头。

```
<?php echo ereg("^ (wang|zhang) +$", "shi and jing"); ?> //false
```

将返回 false，因为待验证的字符串“shi and jing”不满足规则：以“wang”或“zhang”开头。

### 10. 空格字符的处理

空格字符可以简单的处理为普通字符“ ”，但在实际使用中常用“[:space:]”来代替，这样在字符串中更加易读。例如：“I[:space:]am”表示为“I am”。

### 11. 特殊字符的处理

因为一些字符要用在一个正则表达式语法上，像(wang|zhang)中的圆括号，需要屏蔽掉这些字符，使之成为字符串的一部分，而不是具有功能性的表达式的一部分。用转义字符，即反斜杠“\”可以实现这种转换，例如：

```
<?php echo ereg("^ [a-zA-Z] + \ [a-zA-Z] +$", "wang|zhang"); ?>
```

在正则表达式中，需要转义的字符包括：^, \$, (, ), ., [, ], \*, ?, +, \ and { 。

本节全面的介绍了如何构造一个正则表达式，即如何定义一个字符串组成规则。下一节，将给出一系列例子，通过例子，将进一步熟悉上述规则构造方法。

## 3.2.4 示例 1：验证 URL

本小节实现利用 PHP 正则表达式验证 URL 合法性的示例。一个合法的 URL 如：http://www.php.net。其构造规则为：[协议]://[www].[域名].[com|net|org...]

根据上一小节的构造正则表达式，可以构造下面的规则：

```
" ^http://(www\ .)? .+ \ .(com|net|org)$ "
```

其中，“^http://”定义能匹配规则的字符串开头是“http://”；“(www\ .)?”表示随后应该是 0-1 个“www”；而“.+”表示任意字符串；然后是一个“.”，转义字符“\”表明其仅仅是一个字符；最后的“(com|net|org)\$”表明以 com、net、org 中其中一个结尾，此处，只列出这三种情况。完成验证 URL 合法性的函数如下所示。

```
1 <!--使用正则表达式检查 URL 合法性：validate_url.php-->
2 <?php
3 function isValidDomain($domainName)
4 {
5     return (int)ereg("^ (http|ftp) :// (www\ .)? .+ \ . (com|net|org) $", $domainName);
6 }
7 echo isValidDomain("http://www.somesite.com"); //1
8 echo isValidDomain("http://somesite.com"); //1
9 echo isValidDomain("http://www.somesite.fr"); //0
10 echo isValidDomain("www.somesite.com"); //0
11 ?>
```

### 3.2.5 示例 2：验证电话号码

本小节实现利用 PHP 正则表达式验证北京市电话号码合法性的示例。合法的号码如：+86 010xxxxxxxx，其构造规则为：[+86] [010][八位数字]

根据上一小节的构造正则表达式，可以构造下面的规则：

“`^\+86[[:space:]]010[0-9]{8}$`”

其中，“`^\+86`”定义能匹配规则的字符串开头是“+86”；“`[[:space:]]`”表示随后一个空格；而“`[0-9]{8}$`”表明以八个数字结尾。

完成验证北京市电话号码合法性的函数如下所示。

```
1  <!--使用正则表达式检查北京电话号码合法性：validate_phone.php-->
2  <?php
3  function isValidPhone($phoneNum)
4  {
5      echo (int)ereg("^\+86[[:space:]]010[0-9]{8}$", $phoneNum);
6  }
7  echo isValidPhone("+86 01012345678");      //1
8  echo isValidPhone("+86 010123456789");      //0
9  echo isValidPhone("+86 0101234567a");      //0
10 ?>
```

## 3.3 规则数组

同字符串一样，数组也是 PHP 的重要内容，数组能够按一定规律把相关的数据组织在一起，并能快速的管理这些数据。本节和下一节，将深入讨论如何有效地使用数组。

### 3.3.1 理解数组

数组即一组数据，它把一系列数据组织在一起，成为一个可操作的整体。举个简单的例子，当一个做事细心的妻子或丈夫去超市买东西时，他们或许会事先列出一个清单：

- (1) 油
- (2) 盐
- (3) 酱
- (4) 醋
- (5) 毛毛熊
- (6) .....

可以称这个清单为“需购物品”，它规律的列出了其内部的数据，且其内部数据具有相同的性质。在 PHP 中，可以称这样一个清单为数组：

`$list = array(“油”，“盐”，“酱”，“醋”，“毛毛熊”);`

而当使用其中的数据时，因为其每一个数据都对应一个排列中的次序，因此就可以直接利用其这个次序就可得到数据，称这个次序为数组的下标。例如：

```
echo $list[4];
```

即得到“毛毛熊”。

注意：同 C 语言和大部分语言一样，PHP 的下标也是从 0 开始，而不是从 1。因此，下标为 4 的元素指数组的第 5 个元素。

### 3.2.1 定义一个数组

定义一个数组是指定义一个数组的名字和结构，可以初始化其内部数据元素的值，也可以不作初始化处理，即所有的元素值为空。定义数组是使用数组的基础，PHP 提供了 `array()` 函数实现数组的定义，原型如下：

```
array array ( [mixed data ...])
```

系列参数 `data` 可以有多个，用逗号分隔，最后一个数据后无标点，PHP 数组中各数据元素数据类型可以不同，当然也可以是数组类型。当 `data` 为数组类型时，即变成二维数组。

运算符 “`=>`” 用于定义数组元素的值。语法 “`index => values`”，用于定义数组下标和对应的值。下标可以是数字或字符串，字符串下标也称为键。如果省略了下标，会自动产生从 0 开始的整数下标。如果下标是某个整数，则下一个自动产生的下标将是目前最大整数下标 +1。而如果定义了两个完全一样的下标，则后一个会覆盖前面的。下面的示例代码：

```
$array = array( 1, 2, 3, 4, 5, 8=>6, 4=>7, 8, 3=>9);  
print_r($array);
```

将输出：

```
Array ( [0] => 1 [1] => 2 [2] => 3 [3] => 9 [4] => 7 [8] => 6 [9] => 8 )
```

注意下标 3 被定义了两次，保留了最后的值 9。下标 4 在下标 8 之后定义，下一个自动生成的下标（值为 8 那个）为 9，因为之前最大的下标是 8。函数 `print_r()` 常用于输出复杂的数据结构。下面的代码则建立了一个二维数组：

```
$items= array (  
    "食品" => array ("油", "盐", "酱", "醋"),  
    "水果" => array ("a"=>"苹果", "b"=>"桔子", "c"=>"香蕉"),  
    "玩具" => array ("毛毛熊", 5 => "变形金刚", "snoopy")  
);  
print_r($items);
```

将输出：

```
Array (  
    [食品] => Array ( [0] => "油" [1] => "盐" [2] => "酱" [3] => "醋" )  
    [水果] => Array ( [a] => 苹果 [b] => 桔子 [c] => 香蕉 )  
    [玩具] => Array ( [0] => 毛毛熊 [5] => 变形金刚 [6] => snoopy )  
)
```

这是一个二维数组，更高维数的数组定义同二维数组类似。

同 `array()` 函数类似的数组函数还有 `list()`，不同在于 `list()` 只能指定数字下标，并且下标必须从 0 开始。其使用此处不再详述。

### 3.1.2 遍历数组元素

遍历数组中的所有元素是常用的一种操作，在遍历的过程中可以完成查询或其他的功能。例如，想要在购物清单中寻找是否有“拖鞋”这一项，就需要遍历数组操作。PHP 提供 `array_walk()` 函数遍历整个数组，原型如下：

```
bool array_walk ( array array, callback function [, mixed userdata])
```

`array_walk()` 对第一个参数传递过来的数组中的每个元素，执行第二个参数定义的函数 `function()`。典型情况下 `function` 接受两个参数，其中，数组名 `array` 的值为第一个参数，而数组中下标或下标名为第二个参数。如果提供了可选参数 `userdata`，将被作为第三个参数传递给 `function`。函数成功执行返回 `TRUE`，否则返回 `FALSE`。

下面的示例遍历购物清单数组 items，并标识“水果”元素为“:OK”后输出。

```
1  <!--遍历数组：array_walk.php-->
2  <?php
3      $items= array ("a"=>"食品", "b"=>"水果", "c"=>"玩具");    //定义数组
4      function walk(&$my_array, $key) {          //自定义函数，用于改变数组中元素的值
5          if($my_array=="水果")
6              $my_array = "$my_array:OK"; //在元素"水果"的值后面加上“:OK”
7          echo "($key)$my_array<br>";          //输出元素值
8      }
9      array_walk($items,"walk");                //使用 array_walk()遍历整个数组 ,并完成 walk()
函数功能
10 ?>
```

上面示例的执行过程可参考注释部分。示例的输出结果为：

```
(a)食品
(b)水果:OK
(c)玩具
```

如果 function()函数在执行过程中，需要改变数组元素的值，则其参数传递方式应使用引用传递，这样，function()函数对数组所做的操作才能真正引起数组元素值的改变。但是需要注意的是，不能在函数 function 中改变该数组本身的结构，例如增加/删除数据元素等。否则函数 array\_walk()会因其作用的数组会被 function 改变而引起错误。

注意：PHP3.0.3 及以上版本支持 array\_walk()函数，而将键名和 userdata 传递到 function 中则是 PHP 4.0 新增加的。

### 3.1.3 获取当前元素

使用 array\_walk()可以在遍历数组过程中操作每一个元素，很容易想到，可以通过数组下标遍历数组实现相同的功能，并且更容易实现更灵活的操作。PHP 提供了 foreach 语句很方便地实现数组的遍历，原型如下：

foreach(array\_name as \$value) statement

foreach(array\_name as \$key => \$value) statement

foreach 语句类似于 for 循环语句，它获取数组的当前元素，并按下标顺序自动指向下一个元素。上面第一种格式遍历给定的 array\_name 数组，每次循环中，当前元素的值被赋给 \$value，并且数组内部的下标向前移一步（因此下一次循环中将会得到下一个元素）。第二种格式完成同样的功能，区别在于当前元素的键值也会在每次循环中被赋给变量 \$key。

下面示例代码实现如 3.1.2 示例相同的功能：

```
1  <!--遍历数组：foreach.php-->
2  <?php
3      $items= array ("a"=>"食品", "b"=>"水果", "c"=>"玩具");    //定义数组
4      foreach ($items as $key => $value) {
5          if($value=="水果")
6              $value = "$value:OK";          //在元素"水果"的值后面加上“:OK”
7          echo "($key)$value<br>";          //输出元素值
8      }
9  ?>
```

与 foreach 语句具有类似功能的是 each 函数：

array each ( array array\_name)

each()函数返回 array\_name 数组中当前位置的键 / 值对并向前移动。返回值为四个单元的数

组，其下标名为 0, 1, key 和 value。元素 0 和 key 包含数组元素的下标名，1 和 value 包含有数据。当完成对数组的遍历，即下标越界时，each()函数返回 false。下面示例同样完成上例功能：

```
1  <!--遍历数组：each.php-->
2  <?php
3      $items= array ("a"=>"食品", "b"=>"水果", "c"=>"玩具");//定义数组
4      while (list ($key, $value) = each ($items)) {
5          if($value=="水果")
6              $value = "$value:OK";                //在元素"水果"的值后面加上 ":OK "
7              echo "($key)$value<br>";            //输出元素值
8      }
9  ?>
```

### 3.1.4 改变数组大小

在定义数组的时候，可以初始化数组的大小，但在以后的使用中，常常需要改变数组的大小。这好比，在一张 32 开的纸上列出所要购买的物品，但在写清单的时候可能会发现这张纸放不下所要列出的内容，在不希望把清单列在两张纸上的情况下，只能改变纸张的大小。在 PHP 中，扩大数组时必须给对每个元素赋值，可以通过增长下标并赋值的方式扩大数组大小，如：

```
for($i=50; $i <100; $i++) my_array[$i]= " ";
```

更简单的方式是使用 array\_pad()函数，原型如下：

```
array array_pad ( array input, int pad_size, mixed pad_value)
```

函数第一个参数是要操作的数组，第二个参数 pad\_size 是增加后的数组长度，如果 pad\_size 为正，则数组被填补到右侧，如果为负则从左侧开始填补；第三个参数 pad\_value 给出所要增加的数据的值。函数执行后返回被更改以后的 input 数组。参考下面的例子：

```
$input = array (1, 2, 3);
$result = array_pad ($input, 5, 0);          // (1, 2,3, 0, 0)
$result = array_pad ($input, -7, -1);        //(-1, -1, -1, -1, 1, 2,3,)
$result = array_pad ($input, 2, "noop");     //(12, 10, 9)
```

反过来，函数 array\_splice()用于减小数组的大小，原型如下：

```
array array_splice ( array input, int offset [, int length [, array replacement]])
```

第一个参数 input 是要操作的数组；

第二个参数 offset 是要删除掉的数组元素的起始下标，如果为空，则从第一个元素开始，如果 offset 为正，则从 input 数组中该值指定的偏移量开始移除。如果 offset 为负，则从 input 末尾倒数该值指定的偏移量开始移除；

第三个参数 length 指出要删除掉多少元素，如果为空，则删除掉从 offset 到数组的最后一个元素，如果指定了 length 并且为正值，则移除这么多单元。如果指定了 length 并且为负值，则移除从 offset 到数组末尾倒数 length 为止中间所有的单元；

第四个参数 replacement 用于替换被删除的那一部分数组。函数执行后返回被更改以后的 input 数组。参考下面的例子：

```
$input = array ("1", "2", "3", "4");
array_splice ($input, 2);                    //$input = ("1", "2")
$input = array ("1", "2", "3", "4");
array_splice ($input, 1, -1);                 //$input = ("1", "4")
$input = array ("1", "2", "3", "4");
array_splice ($input, 1, count($input), "5"); //$input = ("1", "5")
```

```
$input = array ("1", "2", "3", "4");  
array_splice ($input, -1, 1, array("5", "6")); // $input = ("1", "2", "3", "5", "6")
```

注意：PHP4 及以上版本才支持 array\_pad() 和 array\_splice()。

### 3.1.5 数组求交

两个数组的交是指两者相同元素的集合。多个数组的交可按照左结合律，即从左至右的顺序依次求得。举例来说，假设 my\_array1=(1,2,3)，my\_array2=(2,3,4)，则两者的交为(2,3)。PHP 使用 array\_intersect() 函数得到两个数组的交，原型如下：

array array\_intersect ( array array1, array array2 [, array ...])  
array\_intersect() 返回一个数组，该数组包含了所有在 array1 中也同时出现在所有其他参数数组中的值。注意 array1 中的键名保留不变。参考下例：

```
<?PHP  
$array1 = array ("a" => "1", "2", "3");  
$array2 = array ("b" => "1", "3", "4");  
$result = array_intersect ($array1, $array2);  
?>
```

则 \$result 为：

```
Array  
(  
    [a] => 1  
    [1] => 3  
)
```

注意：PHP4.0.1 及以上版本才支持 array\_intersect()，而 PHP4.0.4 版本使用此函数时有错误。

### 3.1.6 合并两个数组

把两个数组合并在一起是指把一个数组追加到另一个数组中。例如，my\_array1=(1,2,3)，my\_array2=(4,5,6)，则将两者合并将得到(1,2,3, 4,5,6)。函数 array\_merge() 函数完成这个功能，原型如下：

array array\_merge ( array array1, array array2 [, array ...])  
在合并时，如果输入的数组中有相同的字符串键名，则后面的值将覆盖前面的值。然而，如果数组包含数字键名，后面的值将不会覆盖原来的值，而是附加到后面。下面代码：

```
$array1 = array ("name" => "zhang", 33, 24, 32);  
$array2 = array ("a", "b", "name" => "wang", "weight" => "130", 4);  
$result = array_merge ($array1, $array2);  
print_r($result);
```

则输出的结果应该为：

```
Array  
(  
    [name] => wang  
    [0] => 33  
    [1] => 24  
    [2] => 32  
    [3] => a  
)
```

```

[4] => b
[weight] => trapezoid
[5] => 4
)

```

\$array1 中字符串键 “ name ” 的值在合并时被\$array2 中相同键值的 “ wang ” 所覆盖；对于数字键，想\$array2 中的 “ a ” “ b ” 和 “ 4 ”，则会直接合并到\$array1 中，且其下标会自动增长。具有相似功能的函数 array\_merge\_recursive()，区别在于可以保留同时出现在两个数组中相同字符串键值上的元素，如上面例子中的 “ zhang ” 和 “ wang ”。其使用与 array\_merge()相似，不再赘述。

### 3.1.7 反转一个数组

反转数组很容易理解，例如，对于数组 my\_array1=(1,2,3)，其反转后为(3,2,1)。函数 array\_reverse()实现这个功能，原型如下：

```
array array_reverse ( array array [, bool preserve_keys])
```

如果参数 preserve\_keys 为 TRUE 则保留原来的键名；否则，键名和值将同时对应反转。

参考示例：

```

$input = array ("php", 4.0, array ("1", "2"));
$result = array_reverse ($input);
$result_keyed = array_reverse ($input, TRUE);

```

则\$result 为：

```

Array
(
    [0] => Array
        (
            [0] => 1
            [1] => 2
        )
    [1] => 4
    [2] => php
)

```

其键名是在新的数组中重新获取的，而\$result\_keyed 为：

```

Array
(
    [2] => Array
        (
            [0] => 1
            [1] => 2
        )
    [1] => 4
    [0] => php
)

```

仍保留了原数组的键。

注意：只有 PHP4.0.3 及以上版本支持参数 preserve\_keys。



### 3.1.8 获取多个元素

有时，需要获取连续多个元素，例如，在购物清单上，有可能因为资金拮据而需要去掉前 3 个商品。PHP 使用 `array_slice()` 函数一次处理多个数据元素，原型如下：

```
array array_slice ( array array_name, int offset [, int length])
```

函数将返回根据 `offset` 和 `length` 参数所指定的 `array` 数组中的一段序列组成的数组。

如果 `offset` 为正，则序列将从 `array_name` 中 `offset` 开始；如果 `offset` 为负，则序列将从 `array_name` 中距离末端-`offset` 的地方开始。

如果省略 `length`，则序列将从 `offset` 开始一直到 `array` 的末端；如果给出了 `length` 并且为正，则一次取出 `length` 个元素 如果给出了 `length` 并且为负，则序列将终止在距离数组末端-`length` 的地方。函数的使用参考下面示例：

```
$input = array (1, 2, 3, 4, 5);
$output = array_slice ($input, 2);      // $output = 3, 4, 5
$output = array_slice ($input, 2, -1);  // $output = 3, 4
$output = array_slice ($input, -2, 1);  // $output = 4
$output = array_slice ($input, 0, 3);   // $output = 1, 2, 3
```

示例结果参考注释部分。

### 3.1.9 排序数组元素

对整个数据排序是很重要的操作。例如，按照非递减顺序排序数组(5, 3, 2, 4, 1) 将得到(1, 2, 3, 4, 5)；按非递增顺序则得到(5, 4, 3, 2, 1)。`sort()`函数完成排序功能，原型如：

```
void sort ( array array_name [, int sort_flags])：非递减顺序排序数组；
```

```
void rsort ( array array_name [, int sort_flags])：非递增顺序排序数组；
```

```
void usort ( array array_name, callback cmp_function)：使用用户自定义的比较函数  
cmp_function 对数组 array_name 中的值进行排序。
```

`sort()`和 `rsort()`函数的使用非常简单，在 `usort()`的第二个参数的返回值为 0、负数或整数，分别代表等于，小于或大于。下面的示例说明了 `usort()`函数的使用：

```
function cmp ($a, $b) {
    if ($a == $b)    return 0;
    if (($a > $b))    return 1;
    if (($a < $b))    return -1;
}
$my_array = array (3, 2, 5, 6, 1);
usort ($my_array, "cmp");
while (list ($key, $value) = each ($my_array)) {
    echo "($key): $value\n";
}
```

示例结果为：

```
(0): 1
(1): 2
(2): 3
(3): 5
(4): 6
```

其他用于数组排序的函数包括 `usort()`、`uksort()`、`uasort()`、`natsort()`、`krsort()`、`ksort()`，其使用大都与 `usort()`相似，不再一一详述。

### 3.1.10 综合示例

本节将利用前面所讲的字符串和数组的操作技术，实现一个完整的示例：我的书房之图书列表。其功能为按照一定的排列顺序显示我的书房中所有的图书，如图 3.2 所示。



图 3.2 我的书房之图书列表

首先，每本图书的信息用一维数组来表示，包括书名、价格、作者三项，所有的图书信息也放在一个数组中，这样就构成了图书信息二维数组。然后，分别定义比较书名、比较价格、比较出版社的函数，最后利用这些函数对所有书目进行排序并显示出来。

显示图书的页面如下：

```
1 <!--显示图书列表：books_list.php-->
2 <html>
3     <head>
4         <title>我的书房之图书列表</title>
5     </head>
6     <?php
7         //所有图书数组，本例不考虑如何获取图书信息
8         $books_array=array(
9             array( "name"=>"我的 2005",
10                  "price"=>20.00,
11                  "author"=>"wang"),
12             array( "name"=>"家庭烹饪技术",
13                  "price"=>18.23,
14                  "author"=>"zhang"),
15             array( "name"=>"西方哲学史",
16                  "price"=>34.99,
17                  "author"=>"zhou"),
18             array( "name"=>"三侠五义",
19                  "price"=>11.45,
20                  "author"=>"wu"),
21             array( "name"=>"象棋 23 式",
22                  "price"=>22.50,
23                  "author"=>"bao"),
24         );
```

```

25      /*****
26          compare_*():
27          输入：两个数组$array1,$array2;
28          输出：对两个数组*下标的元素进行比较的结果正数、0 或者负数
29      *****/
30      //compare_name
31      function compare_name($array1,$array2){
32          return strcmp($array1[name],$array2[name]);
33      }
34      //compare_price
35      function compare_price($array1,$array2){
36          return ($array1[price]-$array2[price]);
37      }
38      //compare_author
39      function compare_author($array1,$array2){
40          return strcmp($array1[author],$array2[author]);
41      }
42      //在页面上显示图书列表
43      function show_books(&$books_array){
44          if(count($books_array)){
45              foreach($books_array as $key => $value) {
46                  echo "<tr><td>$key</td><td>书 名：$value[name]</td></tr>";
47                  echo "<tr><td></td><td>价 格：$value[price]元</td></tr>";
48                  echo "<tr><td></td><td>作 者：$value[author]</td></tr>";
49              }
50          }
51      }
52      ?>
53      <body>
54      <?php
55          echo "<h2>本书房有书".count($books_array)."本</h2>";
56          switch($_POST["by_what"]){
57              case "by_price":
58                  echo "按价格排序";break;
59              case "by_name":
60                  echo "按书名排序";break;
61              case "by_author":
62                  echo "按作者排序";break;
63          }
64      ?>
65      <table width="600" border=1>
66          <tr>
67              <td width="10%">序号</td>
68              <td width="90%">图书信息</td>
69          </tr>
70          <?php
71              switch($_POST["by_what"]){
72                  case "by_price":
73                      usort($books_array,"compare_price");break;
74                  case "by_name":
75                      usort($books_array,"compare_name");break;
76                  case "by_author":

```

```

78             usort($books_array,"compare_author");break;
79         }
80         show_books($books_array);
81     ?>
82 </table>
83 <form action="<?php echo $PHP_SELF; ?>" method="post">
84     请选择排序方式：
85     <select name="by_what">
86         <option disable>请选择...</option>
87         <option value="by_name">书名</option>
88         <option value="by_price">价格</option>
89         <option value="by_author">作者</option>
90     </select>
91     <input type="submit" name="ok" value="显示">
92 </form>
93 </body>
94 </html>

```

示例在第 83 行使用了 PHP 预定义的变量 `$PHP_SELF`，表示当前文件的绝对路径，包括文件名。

注意：示例第 56 行和第 71 行用到的 `$_POST`，用于获取 HTML 中 FORM 表单中的信息，在 PHP4 以及以前版本支持 `HTTP_POST_VARS` 获取表单内容，然而 PHP5 不再支持，改用 `$_POST` 获取，在实现本示例时，读者请根据所使用 PHP 版本情况自行调整。

## 3.4 相联数组

上一节，介绍了数组的概念以及常用操作，通常意义上的数组也可以称为规则数组，与之对应的是本节所要介绍的相联数组。

### 3.4.1 理解相联数组

相联数组是指键和值具有对应关系的数组。举一个简单的例子，定义一个省会的数组，并希望每个数据元素的键值为其所在的省份：

```

$capital_array=array(
    "Shandong"=>"Jinan",
    "Shanxi"=>"Taiyuan",
    "Hebei"=>"Shijiazhuang",
    "Henna"=>"Zhengzhou"
);

```

在使用该数组时，方便的使用方式是通过键值取得数据元素，例如：

```
$capital ["Shandong"]="Jinan"
```

称这种必须指定关键字和值对应关系的数组为相联数组。相联数组的很多操作同规则数组很相似，区别在于操作时须时刻注意保持关键字和值的相联关系。

### 3.4.2 增加数组元素

向一个相联数组增加一个数据元素，可以直接使用赋值运算符“=”完成，例如，若想向上

一小节中的\$capital\_array 相联数组中增加广州的信息，可以这样实现：

```
$capital_array["Guangdong"]="Guangzhou";
```

注意：如果新增加的键值原来就存在，则信的数据值将会覆盖掉原来的值。

### 3.4.2 删除数组元素

同增加数组元素相比，删除元素稍微复杂一些。PHP 使用 unset()从相联数组中永久的删除一项，原型如下：

```
void unset ( mixed var [, mixed var [, ...]])
```

unset()常用来删除一个变量，也可以用来删除相联数组元素。其使用很简单，其输入参数即为需要删除的变量或关键字，下面代码从数组\$capital\_array 中删除关键字“ Hebei ”及其值，原型如下：

```
unset($capital_array["Hebei"]);
```

一次删除多个元素只需将其用逗号分隔组成参数列表即可。

### 3.4.4 检测键是否存在

检测关键字是指检查一个关键字是否出现在相联数组中，这个操作也是很常用的，比如在新增一个数组元素时，为了避免与原有的关键字冲突，可以先检测其是否已经存在。PHP 可以使用 isset()函数完成这个功能，原型如下：

```
bool isset ( mixed var [, mixed var [, ...]])
```

isset()函数常用来检测一个变量是否赋值或存在，在 2.2.2 小节中曾介绍过。此处，利用其来判定关键字是否存在。举例来说明其使用，在上一小节的\$capital\_array 中检测是否存在键“ Qinghai ”：

```
if( ! isset($capital_array["Qinghai"]))
    $capital_array["Qinghai"]="Xining";
else
    return;
```

注意：isset()函数用来检测变量是否存在，并不检测其值是否为 false、0 或 NULL。如果需要判定是否是这些值，需要使用 empty()函数，后者也在 2.2.2 小节介绍过，不再重述。

### 3.4.5 检测值是否存在

使用 in\_array()函数可以检测一个值是否存在于数组中，原型如下：

```
bool in_array ( mixed needle, array array_name [, bool strict])
```

函数在 array\_name 中搜索 needle，果找到则返回 TRUE，否则返回 FALSE。如果第三个参数 strict 的值为 TRUE 则 in\_array()函数还会检查 needle 的类型是否和 array\_name 中值的类型的相同。下面代码检查数组\$capital\_array 中是否存在值为“ Jinan ”的元素。

```
if(in_array("Jinan",$capital_array))
    echo 1;
else
    echo 0;
```

注意：如果 needle 是字符串，则比较是区分大小写的。并且，在 PHP 版本 4.2.0 之前，needle 不允许是一个数组。

### 3.4.6 小结

相联数组在很多操作上和规则数组相同，如在 3.3 节中所介绍的操作。本节介绍相联数组的概念，通过介绍增加、删除其数据元素操作，旨在强调相联数组中键和价值相互依赖、不可分隔的性质。检测键和价值是否存在是关联数组常用的操作。

## 本章小结

本章主要内容包含两部分：字符串和数组的操作技术。其中，字符串的操作中，正则表达式是重要，也是比较难以理解的内容。在 3.2 节中对其进行了详细的介绍，并通过两个示例来熟悉其使用。数组的操作分为规则数组和相联数组分别介绍，并给出了示例。字符串和数组是 PHP 最常用的数据类型，掌握二者是学习 PHP 编程技术的基础和重要内容。

## 第 13 章 使用 MySQL 管理数据

上一章介绍了如何管理 MySQL，从本章开始，将继续介绍如何使用 MySQL 来管理数据。数据管理主要包括库级操作、表级操作、列级操作，以及数据级操作。

### 13.1 SQL 基础

SQL 语言是 MySQL 服务器能“听懂”的语言，想与 MySQL 交互，就必须首先了解 SQL。在使用诸如 mysql 客户机这样的程序时，就是要发送 SQL 语句给服务器。另外，在实现数据库系统时，也必须熟悉 SQL 语言，通过数据库接口函数，如 mysql\_query() 等，发送 SQL 语句与数据库交互。

#### 13.1.1 SQL 简介

SQL (Structured Query Language, 结构化查询语言) 是关系数据库通用语言，其地位就犹如英语在世界语言中的地位。SQL 已经被 ANSI (美国国家标准化组织) 确定为数据库系统的工业标准。

SQL 的通用性体现在：在不同的数据库管理系统上，用户可以用几乎同样的 SQL 语句执行同样的操作。例如，“select \* from [数据表名]”从某个数据表中取出全部数据，在 Oracle、SQL Server、Foxpro、MySQL、DB2 等关系型数据库中都可以使用。

按照功能，SQL 语言可以分为 4 大类。

- (1) 数据查询语言：查询数据。
- (2) 数据定义语言：建立、删除和修改数据对象。
- (3) 数据操纵语言：完成数据操作的命令，包括查询。
- (4) 数据控制语言：控制对数据库的访问，服务器的关闭、启动等。

常用的 SQL 命令关键字如表 13.1 所示。

表 13.1 常用的 SQL 命令关键字

功能分类	SQL 关键字	功能
数据定义语言	CREATE/ALTER/DROP TABLE	创建/修改/删除表
	CREATE/ ALTER/DROP VIEW	创建/修改/删除视图
	CREATE/ALTER/DROP INDEX	创建/修改/删除索引
数据操纵语言	INSERT	向表中插入新数据行
	UPDATE	更新表中现有的数据
	DELETE	从表中删除几行数据
数据查询语言	SELECT	从一张或多张表中返回数据
数据控制语言	GRANT	为用户赋予特权
	REVOKE	收回用户特权

SQL 语言简单易学、风格统一，利用简单几个英文单词的组合就可以完成所有的功能。在 MySQL 中，可以使用客户机（如 mysql）直接使用 SQL 语言，这些语句几乎可以不加修改地嵌入到前端开发语言中（如 PHP）。利用前端语言的计算能力和 SQL 的数据库操纵能力，

可以快速建立数据库应用程序。

### 13.1.2 了解 SQL 语句

本节通过最常用的 SELECT 语句，讲解几个实例，给读者一个 SQL 的直观印象。SQL 的详细使用将在后续各节逐一介绍。

SELECT 语句从一个或多个数据库表中提取指定的数据，基本语法为：

```
SELECT ColumnName FROM TableName WHERE Condition
```

SQL 忽略空格、制表符，所以可以在语句中添加换行符、制表符和其他空白，使 SQL 语句层次更加清晰。下面是一个有效语句的示例：

```
SELECT sname FROM students
```

其中，FROM 关键字指名数据的来源，可以是表或视图。WHERE 关键字为数据设置一个或多个条件，使 SQL 语句获取更精确的数据。

### 13.1.3 在 SQL 中加注释

MySQL 允许在 SQL 代码中使用注释，这对于把 SQL 存放在文件中很有用处。可用三个方式编写注释。

以“#”号开头直到行尾的所有内容都是注释。

以“--”（注意“--”后还有一个空格）号开头直到行尾的所有内容都是注释。

以“/\*”开始，以“\*/”结束的所有内容都是注释，可注释多行

例如：

```
# This is a single line comment
-- This is a single line comment
/* this is a single line comment*/
/* this is the start of a multiline comment
...
this is the end of a multiline comment*/
```

### 13.1.4 MySQL 中的 SQL 特征

上一小节通过简单的示例，给读者 SQL 的直观印象。作为通用的关系数据库语言，SQL 是各个关系数据库的基础，同时，每个数据库往往都会对其有一些扩展，以支持更为强大的功能。

MySQL 支持的 SQL 语句包括如表 13.2 所示。

表 13.2 MySQL支持的SQL语句

功能	SQL语句
创建、丢弃和选择数据库	CREATE/DROP DATABASE、USE
创建、更改和丢弃表和索引	CREATE/ALTER/DROP TABLE CREATE/ALTER/DROP INDEX
从表中选择信息	SELECT
取数据库、表和查询的有关信息	DESCRIBE、EXPLAIN、SHOW
修改表中信息	DELETE、INSERT、UPDATE、LOAD DATA、OPTIMIZE TABLE、REPLACE



管理语句	FLUSH、GRANT、KILL、REVOKE
其他语句	CREATE/DROP FUNCTION、LOCK/UNLOCK TABLES、SET

同时，MySQL 还对标准的 SQL 语句进行了简化，即它并非支持所有的 SQL 语句所实现的功能，这部分内容将在本章最后一节中给予详细介绍。

## 13.2 创建、删除和选择数据库

上一节介绍了与 MySQL 交互的语言 SQL，从本节开始，就开始介绍使用 MySQL 管理数据的技术。首先，从数据库级的操作开始。

### 13.2.1 创建数据库(CREATE DATABASE)

使用 MySQL 创建数据库很容易，只要在 CREATE DATABASE 语句中给出数据库名称即可：

```
CREATE DATABASE db_name
```

其中，db\_name 是所要创建的数据库名，必须是合法的名称，且不存在同名冲突。同时，登录用户必须有创建数据库的权限。

合法数据库名（以及其他的 MySQL 对象名称）的命名规则如下：

（1）名称可由任意字母、数字、“\_”或“\$”组成，可按上述任意字符起头，但不能单独由数字组成（因为那样会使其与数值相混）。

（2）名称的长度限制为：数据库、表、列和索引的名称最多可由 64 个字符组成，而别名最多可长达 256 个字符。

（3）不能使用 MySQL 的关键字作为数据库、表名。

举例：使用 mysql 客户机，下面语句创建并查看数据库 student\_course：

```
mysql>CREATE DATABASE student_course;
mysql> show databases;
+-----+
| Database |
+-----+
| mysql    |
| student_course |
+-----+
2 rows in set (0.07 sec)
```

### 13.2.2 删除数据库(DROP DATABASE)

删除数据库使用 DROP DATABASE 语句：

```
DROP DATABASE db_name
```

DROP DATABASE 语句将会删除数据库及其所有的表，应谨慎使用该语句。在删除了一个数据库后，该数据库就永远没有了。如果管理员已经正常完成了数据库备份，那么删除的数据库可能还可以恢复。

举例：使用 mysql 客户机，下面语句删除数据库 student\_course：

```
mysql>DROP DATABASE student_course;
mysql> show databases;
+-----+
| Database |
```

```
+-----+
| mysql |
+-----+
1 rows in set (0.07 sec)
```

注意：数据库是由数据目录中的一个目录表示的，如果在该目录中放置了一些非表的数据文件，它们是不会被 DROP DATABASE 语句删除的。此时，该数据库目录自身也不被删除。

### 13.2.3 选择数据库(USE)

USE 语句选择一个数据库，使其成为（当前）缺省数据库：

```
USE db_name
```

选择缺省数据库并不代表在连接期间它都是缺省的。只要具有使用数据库的权限，就可随时提交 USE 语句切换当前数据库。选择一个数据库也不限制只使用该数据库中的表，可以利用数据库作为前缀，使用其他数据库中的表。

例如，当前缺省数据库为 db1，则下面命令也可以查询数据库 db2 中的数据：

```
SELECT * FROM db2.table1
```

这相当于：

```
USE db2
SELECT * FROM table1
```

在关闭服务器时，服务器将不保留缺省数据库的信息。即当再次连接到服务器时，它不会记住以前所选择的数据库。

## 13.3 创建、修改、删除数据表

上一节介绍了数据库级的操作，本节继续介绍 MySQL 数据表级的操作，包括创建、修改、删除表。

### 13.3.1 创建表(CREATE TABLE)

MySQL 使用 CREATE TABLE 语句创建表，简单语法为：

```
CREATE TABLE table_name
(
    column_name_1 data_type_1 constraint_1,
    column_name_2 data_type_2 constraint_2,
    ...
    column_name_n data_type_n constraint_n
)
```

其中，column\_name 为表中的列名，data\_type 为列的数据类型，constraint 为该列的限制说明。CREATE TABLE 其完整语法是相当复杂的，因为存在很多的可选子句：

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] table_name
    (create_definition,...)
    [table_options]
    [[IGNORE|REPLACE] select_statement]
```

```

create_definition=
{
    column_declaration
    |PRIMARY KEY (index_columns)
    |KEY [index_name] (index_columns)
    |INDEX [index_name](index_columns)
    |UNIQUE [INDEX] [index_name] (index_columns)
    |[CONSTRAINT symbol] FOREIGN KEY index_name (index_columns)
    |[reference_definition]
    |CHECK (expression)
}

column_declaration=
    column_name column_data_type
        [NOT NULL|NULL] [DEFAULT default_value]
        [AUTO_INCREMENT] [PRIMARY KEY] [reference_definition]

reference_definition=
    REFERENCES table_name [(index_columns)]
        [MATCH FULL|MATCH PARTIAL]
        [ON DELETE refenece_option]
        [ON UPDATE refenece_option]

refenece_option=
    {RESTRICT|CASCADE|SET NULL|NO ACTION|SET DEFAULT}

```

其中，create\_definition 可以是列定义（column\_declaration）、索引定义、FOREIGN KEY 子句、reference\_definition 或 CHECK 子句。

## 1. 列定义

列定义（column\_declaration）以列名（column\_name）和数据类型（column\_type）开始，后面可跟几个可选的关键字。其中，列类型可以是第 10.4 节列出的任意类型，可跟在列类型之后的可选关键字包括：

**NULL 或 NOT NULL：**指出该列是否可以包含 NULL 值。如果两者都不指出，缺省为 NULL。

**DEFAULT default\_value：**给出该列的默认值，不能用于 BLOB 或 TEXT 列类型。如果无缺省说明，自动分配一个默认值。对于可取 NULL 值的列，默认值为 NULL。对于不能为 NULL 的列，默认值如下：

对于数值列：对于 AUTO\_INCREMENT 列，其默认值为该列的序列中的下一个数。否则，默认值为 0。

日期和时间列：对于 TIMESTAMP 类型，默认值为当前日期和时间。否则，默认值为该类型的“零”值（例如，DATE 类型的“零”值为“0000-00-00”）。

字符串类型：对于非 ENUM 列，默认值为空串。对于 ENUM 列，默认值为第一个枚举元素。

**AUTO\_INCREMENT：**仅用于整数列类型，使这一列上的数值自动增长。初始值可用 AUTO\_INCREMENT 选项明确指定，缺省为 1。AUTO\_INCREMENT 列还必须指定为 UNIQUE 索引或 PRIMARY KEY，而且应该为 NOT NULL。

**PRIMARY KEY：**指定该列为主键。

**UNIQUE：**指定该列为 UNIQUE 索引列。

## 2. 索引定义

PRIMARY KEY、UNIQUE、INDEX 和 KEY 子句定义索引。其中 PRIMARY KEY 与 UNIQUE 指定值必须唯一的索引。INDEX 和 KEY 意义相同，指定可以包含重复值的索引。索引基于 index\_columns 中所指定的列，如果有多个列，用逗号将它们分隔。如果未给出索引名 index\_name，系统会根据第一个索引列的名称自动选一个。

下面的 SQL 语句建立一个含有三列的 students 表，在 id 列上创建主键索引，并在 (name, age) 列上创建 INDEX 索引：

```
CREATE TABLE students
(
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    age INT NOT NULL,
    PRIMARY KEY (id),
    INDEX(name,age)
)
```

### 13.3.2 创建不存在的表(IF NOT EXISTS)

在创建表时，有时候会因为已经同名冲突而创建失败。这时，需要事先做一个判断，看看是否存在同名的表，使用 CREATE TABLE IF NOT EXISTS 即可达到这个目的。

在事先不能判断是否存在重名冲突的时候，使用 IF NOT EXISTS 对于批量 SQL 程序脚本极为有用。如果使用普通的 CREATE TABLE 语句，程序第一次运行时，建立这些表，而第二次运行时将出错。如果用 IF NOT EXISTS 语句，就不会有问题：在第二次运行时，创建表失败，但不出错，使程序可以继续向下运行。

下例中，首先检查是否存在表 students，如果不存在，则创建之，否则，放弃创建：

```
CREATE TABLE IF NOT EXISTS students
(
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    age INT NOT NULL,
    PRIMARY KEY (id),
    INDEX(name,age)
)
```

### 13.3.3 创建临时表(TEMPORARY)

有时，需要临时创建一个中间表，完成一些临时存储数据的功能，在完成临时功能之后，再删除这些临时表。MySQL 可用 CREATE TEMPORARY TABLE 来创建临时表，这些表在与服务器的交互结束时会自动删除，而且如果与服务器的交互异常结束，这些表仍会被删除。

举例：下面的 SQL 语句创建一个临时表 temp\_student：

```
CREATE TEMPORARY TABLE temp_student
(
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name VARCHAR(30) NOT NULL,
    age INT NOT NULL,
    PRIMARY KEY (id),
    INDEX(name,age)
)
```

)

### 13.3.4 利用 SELECT 的结果创建表

在关系数据库中，任何数据都表示为行和列组成的二维表，而每条 SELECT 语句的结果也都是一个二维表。考虑到这种性质，有时，可以把查询结果构造为一个表，把查询数据存储起来，例如：

```
CREATE TABLE new_table_name SELECT * FROM talbe_name [WHERE ...]
```

举例：下面的 SQL 语句创建一个表 young\_students，内容为 students 中年龄小于 20 岁的学生：

```
CREATE TABLE young_students SELECT * FROM students WHERE age <20
```

#### 1. 创建空表

另外，可以通过选择表的全部内容（无 WHERE 子句）来完全拷贝表，也可以利用一个总是失败的 WHERE 子句（如：“WHERE 1<>1”）来创建一个空表。如果希望利用 LOAD DATA 语句将数据文件装入一个表中，就需要首先创建一个具有相同结构的空表，如：

```
CREATE TABLE new_table_name SELECT * FROM talbe_name WHERE 1<>1
```

### 13.3.5 修改表(ALTER TABLE)

修改表指修改表的结构，当发现某个表的结构不满足要求时，可以用 ALTER TABLE 语句来修改表中列的属性，甚至可以修改表的名称。例如，原来的 students 表中没有“性别”属性，或者是“姓名”属性长度不够等等，这些问题都可以通过修改表解决。

ALTER TABLE 的语法如下：

```
ALTER [IGNORE] TABLE table_name action_list
```

MySQL 扩充了 ALTER TABLE 语句，允许指定多个动作，各动作间以逗号分隔，每个 action 表示对表的一个修改。一方面，这方便多个修改操作；另一方面，这种方式能同时将所有 VARCHAR 列更改为 CHAR 列，从而实现将表从行可变长的表更改为行定长的表。

下面的例子给出了某些 ALTER TABLE 功能：

#### 1. 对表重新命名

这很简单，只需给出旧表名和新表名即可：

```
ALTER TABLE table_name RENAME AS new_table_name
```

#### 2. 更改列类型

为了更改列的类型，可使用 CHANGE 或 MODIFY 子句。

举例：假如表 students 中的列 id 为 SMALLINT UNSIGNED 的，希望将其更改为 INT UNSIGNED 的列：

```
ALTER TABLE students MODIFY id INT UNSIGNED
```

```
ALTER TABLE students CHANGE id id INT UNSIGNED
```

在 CHANGE 命令中列名 id 出现了两次，这是因为：CHANGE 除了更改类型外还能更改列名，而 MODIFY 不能实现这个功能。如果希望在更改类型的同时重新将 id 命名为 sid，可按如下进行：

```
ALTER TABLE students CHANGE id sid INT UNSIGNED
```

注意：使用 CHANGE 更改了列的定义，并说明了一个包括列名的列的完整定义，即使不更改列名，也需要在定义中包括相应的列名。

### 3. 将字符串列从可变长转换为定长

对字符串列来说，定长的列一般比变长的列处理更快。例如，students 中列 name 的类型为 VARCHAR，为了提高查询效率，需要转换为定长的 CHAR 列，可以如下修改：

```
ALTER TABLE students MODIFY name CHAR(40)
```

### 4. 将字符串列从定长转换为可变长

虽然字符串列用定长行查询效率更快，但要占用更多的空间，使用可变长度得字符串列可以节省存储空间。这种转换更为容易：只需将某个 CHAR 列转换为 VARCHAR 列，MySQL 就自动地转换其他的 CHAR 列：

```
ALTER TABLE students MODIFY name VARCHAR(40)
```

其他对表结构的修改选项如上面所给出的示例相似，如果想详细了解 action\_list 参数所有的操作，可以参考 13.3.1 中 CREATE TABLE 中 create\_definition 参数的说明。

## 13.3.6 删除表(DROP TABLE)

删除表要比创建和修改表要容易得多，只需指定表名即可：

```
DROP TABLE table_name
```

MySQL 对 DROP TABLE 语句在某些有用的方面做了扩充。首先，可在同一语句中指定几个表对它们进行删除，如：

```
DROP TABLE students,courses,...
```

其次，如果不能肯定一个表是否存在，如果存在就删除它，否则也不会出现错误，那么可在 DROPTABEL 语句中增加 IF EXISTS。如：

```
DROP TABLE IF EXISTS students
```

同 CREATE TABLE 一样，IF EXISTS 语句在含有 DROP TABLE 的 SQL 脚本中很有用，如果不存在待删除的表，脚本会继续向下执行而不会抛出错误。

## 13.4 创建和删除索引

索引是加速查询的主要手段，特别对于涉及多个表的查询更是如此。本节中，将介绍索引的作用、特点，以及创建和删除索引的语法。

### 13.4.1 使用索引优化查询

索引是快速定位数据的技术，首先通过一个示例来了解其含义及作用，详细的介绍请看考第 14 章。

#### 1. 索引示例

假设对于 10.3 节所建的表，各个表上都没有索引，数据的排列也没有规律，如表 13.4 所示。

表 13.3 没有索引的students表

sid	sname	sgender	sage
52	zhang	M	21

22	wang	M	22
33	li	F	19
41	zhao	M	20
...	...	...	

当查找某个学生信息时，必须顺序查看表 students 中的每一行，检查是否与所需值匹配，这需要扫描全表，效率很低。

表 14.2 给出了在 name 列上增加了索引的 students 表。

表 12.2 在name列上增加了索引的students表

Index		Sid	Sname	sgender	sage
l	→	52	Zhang	M	21
w	→	22	wang	M	22
S	→	33	li	F	19
z	→	41	zhao	M	20
		...	...	...	

索引是在 name 上排序的。现在，当查找某个学生信息时，就不需要逐行搜索全表，而是可以利用索引进行有序查找（如二分查找法），快速定位到匹配的值，以节省大量搜索时间。

## 2. 索引作用

在索引列上，除了上面提到的有序查找之外，数据库利用各种各样的快速定位技术，能够大大提高查询效率。特别是当数据量非常大，查询涉及多个表时，使用索引往往能使查询速度加快成千上万倍。

例如，有三个未索引的表 t1、t2、t3，分别只包含列 c1、c2、c3，每个表分别含有 1000 行数据组成，指为从 1 ~ 1000 的数值，查找对应值相等行的查询如下所示：

```
SELECT c1,c2,c3 FROM t1,t2,t3 WHERE c1=c2 AND c1=c3
```

此查询结果应该为 1000 行，每行包含 3 个相等的值。在无索引的情况下处理此查询，必须寻找三个表所有的组合，以便得出与 WHERE 子句相配的那些行。而可能的组合数目为  $1000 \times 1000 \times 1000$ （十亿），显然查询将会非常慢。

如果对每个表进行索引，就能极大地加速查询进程。利用索引的查询处理如下：

- （1）从表 t1 中选择第一行，查看此行所包含的数据；
- （2）使用表 t2 上的索引，直接定位 t2 中与 t1 的值匹配的行。类似，利用表 t3 上的索引，直接定位 t3 中与来自 t1 的值匹配的行；
- （3）进到表 t1 的下一行并重复前面的过程，直到遍历 t1 中所有的行。

在此情形下，仍然对表 t1 执行了一个完全扫描，但能够在表 t2 和 t3 上进行索引查找直接取出这些表中的行，比未用索引时要快一百万倍。

利用索引，MySQL 加速了 WHERE 子句满足条件行的搜索，而在多表连接查询时，在执行连接时加快了与其他表中的行匹配的速度。

## 13.4.2 创建索引

在执行 CREATE TABLE 语句时可以创建索引，也可以单独用 CREATE INDEX 或 ALTER TABLE 来为表增加索引。

### 1. ALTER TABLE

ALTER TABLE 用来创建普通索引、UNIQUE 索引或 PRIMARY KEY 索引，如：

```
ALTER TABLE table_name ADD INDEX index_name (column_list)
ALTER TABLE table_name ADD UNIQUE (column_list)
ALTER TABLE table_name ADD PRIMARY KEY (column_list)
```

其中 table\_name 是要增加索引的表名, column\_list 指出对哪些列进行索引, 多列时各列之间用逗号分隔。索引名 index\_name 可选, 缺省时, MySQL 将根据第一个索引列赋一个名称。另外, ALTER TABLE 允许在单个语句中更改多个表, 因此可以在同时创建多个索引。

## 2 . CREATE INDEX

CREATE INDEX 可对表增加普通索引或 UNIQUE 索引, 如:

```
CREATE INDEX index_name ON table_name (column_list)
CREATE UNIQUE INDEX index_name ON table_name (column_list)
```

table\_name、index\_name 和 column\_list 具有与 ALTER TABLE 语句中相同的含义, 索引名不可选。另外, 不能用 CREATE INDEX 语句创建 PRIMARY KEY 索引。

## 3 . 索引类型

在创建索引时, 可以规定索引能否包含重复值。如果不包含, 则索引应该创建为 PRIMARY KEY 或 UNIQUE 索引。对于单列惟一性索引, 这保证单列不包含重复的值。对于多列惟一性索引, 保证多个值的组合不重复。

PRIMARY KEY 索引和 UNIQUE 索引非常类似。事实上, PRIMARY KEY 索引仅是一个具有名称 PRIMARY 的 UNIQUE 索引。这表示一个表只能包含一个 PRIMARY KEY, 因为一个表中不可能具有两个同名的索引。

下面的 SQL 语句对 students 表在 sid 上添加 PRIMARY KEY 索引:

```
ALTER TABLE students ADD PRIMARY KEY (sid)
```

### 13.4.3 删除索引

可利用 ALTER TABLE 或 DROP INDEX 语句来删除索引。类似于 CREATE INDEX 语句, DROP INDEX 可以在 ALTER TABLE 内部作为一条语句处理, 语法如下:

```
DROP INDEX index_name ON table_name
ALTER TABLE table_name DROP INDEX index_name
ALTER TABLE table_name DROP PRIMARY KEY
```

其中, 前两条语句是等价的, 删除掉 table\_name 中的索引 index\_name。

第三条语句只在删除 PRIMARY KEY 索引时使用, 因为一个表只可能有一个 PRIMARY KEY 索引, 因此不需要指定索引名。如果没有创建 PRIMARY KEY 索引, 但表具有一个或多个 UNIQUE 索引, 则 MySQL 将删除第一个 UNIQUE 索引。

如果从表中删除了某列, 则索引会受到影响。对于多列组合的索引, 如果删除其中的某, 则该列也会从索引中删除。如果删除组成索引的所有列, 则整个索引将被删除。

## 13.5 增删改数据

前几节介绍了库级及表级的操作, 却并未真正涉及到数据的操作。本节就将介绍如何在数据表中增加、修改、删除记录。本节仍然使用在 10.3 节所建立的 students\_courses 数据库, 包含 students、courses、stu\_cou 三个表, 分别表示学生、课程, 和学生 - 成绩关系。

### 13.5.1 使用 INSERT 增加记录

向表中增加新记录有几种方法:



通过 INSERT 语句将记录插入表中；

通过从某个文件读取数据来增加记录，可以是利用 LOAD DATA 或 mysqlimport 实用程序装入数据，也可以是预先写成多个 INSERT 语句的形式。

本节将首先介绍使用 INSERT 语句。使用 INSERT 语句，需要指定接收数据的表，以及所要增加的数据，具有几种形式：

### 1. 给出所有列的值

语法如下：

```
INSERT INTO table_name VALUES(value1,value2,...)
```

例如：

```
#向 students 表中添加两个学生记录张三
mysql> INSERT INTO students VALUES
-> (1,'张三','男',21);
#向 courses 表中添加三个课程记录 C 语言
mysql> INSERT INTO courses VALUES
-> (1,'C 语言',3,'李老师');
#向 stu_cou 表中添加张三和李四的成绩信息
mysql> INSERT INTO stu_cou VALUES
-> ( 1,2,90 );
```

其中，VALUES 必须包含表中每列的值，并且按表中列的存放次序以此给出。在 MySQL 中，需要用单引号或双引号将串和日期值括起来。如果某一列为 AUTO\_INCREMENT，则可赋值为 NULL。

另外，利用单个的 INSERT 语句可以一次将几行插入一个表中：

```
INSERT INTO table_name VALUES (...),(...),...
```

如下所示：

```
mysql> INSERT INTO students VALUES
-> (1,'张三','男',21),
-> (2,'李四','男',22);
```

### 2. 给出要赋值的列，然后再给出值

语法如下：

```
INSERT INTO table_name(column_name1,column_name2,...) VALUES(value1,value1,...)
```

这种方式适用于只给表中的某些列赋值，例如：

```
mysql> INSERT INTO courses(cid,cname,ccredit) VALUES
-> (1,'C 语言',3);
```

同样，也可以一次添加多个记录：

```
mysql> INSERT INTO courses(cid,cname,ccredit) VALUES
-> (1,'C 语言',3);
-> (2,'数据结构',2);
```

对于未指定的列，所添加的指为默认值，或 NULL。

### 3. 用 col\_name = value 的形式给出列和值

语法如下：

```
INSERT INTO table_name SET column_name1=value1,column_name2=value2,...
```

例如：

```
mysql> INSERT INTO courses SET
-> (cid=1,,cname='C 语言',ccredit=3);
```

使用这种形式的 INSERT 不能一次插入多行数据。

### 13.5.2 使用 LOAD DATA 批量增加记录

从文件读取数据可以将大量记录装载到表中。LOAD DATA 语句可以批量装载数据，它从一个文件中读取数据：

```
mysql> LOAD DATA LOCAL INFILE "students_data.txt" INTO TABLE students;
```

该语句读取客户机当前目录中数据文件 students\_data.txt 的内容，并将其发送到服务器并装入 students 表。缺省时，LOAD DATA 语句假定列值由 tab 键分隔，而行以换行符结束，并假定各个值是按列在表中的存放次序给出的。

### 13.5.3 使用 mysqlimport 批量增加记录

除 LOAD DATA 外，还可以使用 mysqlimport 实用程序直接从文件读取批量数据。实际上，mysqlimport 实用程序相当于 LOAD DATA 的命令行的一个接口。

从外壳程序调用 mysqlimport：

```
% mysqlimport --local students_courses student.txt
```

Mysqlimport 自动生成一个 LOAD DATA 语句，此语句把 student.txt 文件中的数据装入 student 表。Mysqlimport 根据数据文件名导出表名（将文件名第一个圆点前的所有字符作为表名）。例如，student.txt 将被装入 student 表，而 president.txt 将被装入 president 表。如果有多个需要装入单个表的文件，应仔细地确定文件名，使表名一致，如 table.1.txt、table.2.txt 等，否则 mysqlimport 将不能使用正确的表名。

对于如像 table1.txt 与 table2.txt 这样的文件名，mysqlimport 将会认为相应的表名为 table1 和 table2。

### 13.5.4 修改记录（UPDATE）

为了修改现有记录，可利用 UPDATE 语句，格式如下：

```
UPDATE table_name  
SET column_name1=new_value1,column_name2=new_value2,...  
WHERE Condition
```

WHERE 子句是可选的，指定所要修改的记录。如果不指定的话，表中的每个记录都被更新。下面的查询将每个学生的性别都更改为“F”：

```
UPDATE students SET sgender='F'
```

将每个学生的年龄都加一：

```
UPDATE students SET sage=sage+1
```

本节，将不详细介绍 WHERE 子句，而是在下一节“查询数据”中详细介绍。

### 13.5.5 删除记录（DELETE）

DELETE 语句用于现有记录，格式如下：

```
DELETE FROM table_name  
WHERE Condition
```

同 UPDATE 一样，WHERE 子句是可选的，指定所要删除的记录。如果不指定，表中的每

个记录都被删除。下面的查询将删除性别为“F”的学生：

```
DELETE FROM students WHERE sgender='f'
```

下面的查询将删除性别为“F”并且年龄大于21的学生：

```
DELETE FROM students WHERE sgender='f' AND sage>21
```

而不带 WHERE 的 DELETE 查询将删除所有的学生：

```
DELETE FROM students
```

同样，本节不详细介绍 WHERE 字句。

## 13.6 查询数据

上一节使用 INSERT、UPDATE、DELETE 语句向 students\_courses 数据库的表中增删改了一些数据，本节将继续介绍更为重要的操作：使用 SELECT 查询数据。SELECT 是最常用的语句，而且使用也最为讲究，语法如下：

```
SELECT selection_list -- 选择哪些列
FROM table_list -- 从何处选择行
WHERE primary_constraint -- 行必须满足什么条件
GROUP BY grouping_columns -- 怎样对结果分组
ORDER BY sorting_columns -- 怎样对结果排序
HAVING secondary_constraint -- 行必须满足的第二条件
LIMIT count 结果限定
```

除了“SELECT”和说明检索内容的 column\_list 外，其他项都是可选的。

### 13.6.1 选择列

在使用一个 SELECT 语句时，首先需要确定所要查询的列，可以使用“\*”来表示“所有列”。

例如，下面的查询检索 students 表中所有数据：

```
mysql> select * from students;
+----+-----+-----+-----+
| sid | sname  | sgender | sage |
+----+-----+-----+-----+
| 1 | 张三  | 男     | 21  |
| 2 | 李四  | 男     | 22  |
| . | ....  | ..     | ...  |
+----+-----+-----+-----+
x rows in set (0.04 sec)
```

各列按在表中存放的次序以此出现，该次序与执行 DESCRIBE student 语句时显示的列次序相同，也可明确地指定希望得到的一列或多列。例如，如果只选择学生名，提交下列语句：

```
mysql> select sname from students;
+-----+
| sname |
+-----+
| 张三  |
| 李四  |
| ....  |
+-----+
```

如果名字不止一列，可用逗号分隔它们。下列的语句与 SELECT \* FROM student 等价，只

是明确地指出了每一列：

```
mysql> select sid,sname,sgender,sage from students;
```

可按任意次序给出列：

```
mysql> select sname,sgender,sage,sid from students;
```

```
+-----+-----+-----+-----+
| sname | sgender | sage | sid |
+-----+-----+-----+-----+
| 张三  | 男      | 21   | 1   |
| 李四  | 男      | 22   | 2   |
| . | ...    | ..   | ... |
+-----+-----+-----+-----+
```

对于取出的列，还可以给它起一个别名，如：

```
mysql> select sname as name,sgender as gender,sage as age,sid as id from students;
```

```
+-----+-----+-----+-----+
| name | gender | age | id |
+-----+-----+-----+-----+
| 张三 | 男      | 21  | 1  |
| 李四 | 男      | 22  | 2  |
| . | ...    | ..   | ... |
+-----+-----+-----+-----+
```

注意：列名在 MySQL 中不区分大小写。

## 13.6.2 选择数据表

在使用 SELECT 语句时，需要确定所要查询数据的所在的表，若数据存在于多个表中，这时需要使用逗号将多个表名分隔开。

### 1. 从一个表中获取数据

首先来看从一个表中获取数据的情况，这种情况非常简单，只需要把表名放在 FROM 关键字之后即可。例如，下面的 SQL 获取 students 表中年龄大于 21 的学生信息：

```
mysql> select * from students where sage >21;
```

```
+-----+-----+-----+-----+
| sid | sname | sgender | sage |
+-----+-----+-----+-----+
| 2 | 李四 | 男 | 22 |
+-----+-----+-----+-----+
```

### 2. 从多个表中获取数据

如果数据存在于多个表中，则需要把这几个表都列在 FROM 关键字之后，并用逗号分隔。此时，WHERE 子句常常需要通过连接运算来确定多个表的关联关系。例如，下面的 SQL 获取所有成绩不及格的学生名字：

```
mysql> SELECT sname FROM students,stu_cou
-> WHERE
-> students.sid=stu_cou.sid AND stu_cou.score<60
-> ;
```

其中，students.sid=stu\_cou.sid 将表 students 和 stu\_cou 连接起来，称为等同连接。如果不用 students.sid=stu\_cou.sid，则产生的结果将是两个表的笛卡尔集，称为全连接。

#### 全连接

全连接也称为交叉连接，每个表的每行都与其他表中的所有行连在一起，以产生所有可能的组合，也就是笛卡儿积，连接表的行数为每个表的行数之积。例如，三个分别含有 100、200、300 行的表的全连接，将产生  $100 \times 200 \times 300 = 6'000'000$  行数据。即使各表很小，所得到的

行数也会很大。在这样的情形下，通常要使用 WHERE 子句来将减少结果集。

#### 等同连接

如果在 WHERE 子句中增加一个条件，使各表在某些列相等的行进行组合，这就是的等同连接，只选择那些在指定列中具有相等的值的行。

另外，有的多个表中具有相同名的列名，这时在查询中需要用 tablename.colname 来指明列具体属于哪个表中。例如，下面的 SQL 语句指定 sname 属于 students 表：

```
mysql> SELECT students.sname FROM students,stu_cou
-> WHERE
-> students.sid=stu_cou.sid AND stu_cou.score<60
->;
```

如果不存在重名的列，可以省略指定表名。

### 13.6.3 使用 WHERE 选择记录

使用 WHERE 子句，可给出数据的查询条件（Condition）。常见的查询条件逻辑运算符如表 13.5 所示。

表 13.5 where条件字句中常用的逻辑运算符

运算符	意义
=	等于
LIKE	类似于（支持通配符）
<>	不等于
NOT	LIKE不类似于（支持通配符）
<	小于
>	大于
<=	小于或等于
>=	大于或等于
AND	两个条件都必须满足，如：上海AND北京
OR	至少必须满足其中一个条件，如：老张OR老王
NOT	排除后跟的条件，如：巴黎NOT法国

通常可以使用下面的运算来指定所要获取的数据所满足的条件：

#### 1．根据数字值确定条件

例如：查询所有及格的学生：

```
mysql> SELECT * FROM stu_cou
> WHERE scORe > 60;
```

#### 2．根据字符串串值确定条件

例如：查询所有女学生：

```
mysql> SELECT * FROM students
> WHERE sgender ='f';
```

查询所有姓张学生：

```
mysql> SELECT * FROM students
> WHERE sname like '张%';
```

#### 3．根据日期型值确定条件

例如：查询所有出生日期小于 1981 年的学生（原表中没有 sbirthday 项，请读者自行 ALTER TABLE）

```
mysql> SELECT * FROM students
> WHERE sbirthday <'1981-01-01';
```

#### 4. 根据 NULL 值确定条件

NULL 值是特殊的，它代表“无值”，不能用处理已知值的相同方式来处理 NULL。如果试图与通常的算术、比较运算符一起使用 NULL，其结果是 NULL。同样，因为不可能知道比较两个未知值的结果，不能将 NULL 与它自身比较。例如：

```
mysql> SELECT NULL=NULL, NULL=0;
```

NULL=NULL	NULL=0
NULL	NULL

为了进行 NULL 值的搜索，必须采用特殊的语法。不能用= 或<>来判断等于 NULL 或不等于 NULL，取而代之的是使用 IS NULL 或 IS NOT NULL。

例如：查找所有任课老师不空的课程

```
mysql> SELECT * FROM courses  
-> WHERE cteacher IS NOT NULL;
```

#### 5. 综合条件

例如：查询所有出生日期小于 1981 年、成绩不及格的的女学生

```
mysql> SELECT * FROM students,stu_cou  
-> WHERE students.sid=students.sid  
-> AND sbirthday <'1981-01-01' AND sgender ='f' AND score < 60;
```

查询所有出生日期小于 21 岁的女学生，和小于 22 岁的男学生

```
mysql> SELECT * FROM students  
-> WHERE (sage<21 AND sgender='f') OR (sage<22 AND sgender='m');
```

### 13.6.4 使用 GROUP BY 对结果分组

有时候，需要对查询数据首先进行分组，然后做一些操作。例如，想获取所有男生和女生的平均年龄。有时甚至不知道数据会分为几组，如想获取按不同年龄分组后的平均成绩。这时，就需要使用 GROUP BY 子句时。

示例：查询按性别分组后的平均年龄：

```
mysql> SELECT AVG(sage) as avg_sage,sgender FROM students GROUP BY sgender ;
```

AVG_sage	sgender
21.5000	男
20.0000	女

示例：查询不同年龄学生的平均成绩：

```
mysql> SELECT sage,AVG(score) FROM students,stu_cou  
-> where students.sid=stu_cou.sid  
-> GROUP BY sage;
```

sage	AVG(score)
21	89.0000
22	78.0000

在使用 GROUP BY 分组时，所查询的列必须包含在分组的列中，或者在 AVG()或 SUM()函

数中。这样做的目的是，使查询到的数据没有矛盾。看下面的例子：

```
mysql> SELECT sname,AVG(sage) as avg_sage,sgender FROM students GROUP BY sgender
;
+-----+
| sname | AVG_sage | sgender |
+-----+
| 张三  | 21.5000  | 男      |
+-----+
```

这时，虽然 MySQL 没有出错，但是值 sname 为张三是没有什么意义的。结合上面的查询结果，请读者自己考虑其原因。

### 13.6.5 使用 DISTINCT 对结果去重

利用 DISTINCT 关键字，可以删除结果中的重复行。例如，下面的 SQL 从 courses 表中获取所有的教师信息：

```
mysql> SELECT DISTINCT cteacher from courses ;
```

而不会出现重复的记录，即如果有一个教师教多门课，查询结果中也只出现一次。

### 13.6.6 使用 ORDER BY 对结果排序

有时，需要按一定的顺序获取数据，比如，想按姓名或者学号顺序得到所有学生的信息。这时，可以使用 ORDER BY 子句来实现结果的排序。

例如，按姓名先后顺序得到学生信息：

```
mysql> SELECT * FROM students ORDER BY sname;
+-----+
| sid | sname | sgender | sage |
+-----+
| 2   | 李四  | 男      | 22   |
| 1   | 张三  | 男      | 21   |
...
+-----+
3 rows in set (0.00 sec)
```

默认情况下，ORDER BY 按升序给出结果，如果想按降序得到结果，可以使用 DESC 关键字。例如：按数据结构成绩从高到低得到学生信息：

```
mysql> SELECT sname FROM students,stu_cou,courses
-> WHERE students.sid=stu_cou.sid AND courses.cid=stu_cou.cid
-> AND courses.cname='数据结构'
-> ORDER BY score DESC;
+-----+
| sname |
+-----+
| 李四  |
| 张三  |
...
+-----+
3 rows in set (0.00 sec)
```

在对包含 NULL 值的列进行排序时，如果是升序排序，NULL 值出现在最前面，如果是按

降序排序，NULL 值出现在最后。

可在多个列上进行排序，而每个列的升序或降序可以互相独立。例如，对课程表，首先根据教师姓名排序，然后根据其课程名按逆序排序：

```
mysql> SELECT * FROM courses
-> ORDER BY cteacher, cname DESC;
+-----+-----+-----+-----+
| cid | cname    | ccredit | cteacher |
+-----+-----+-----+-----+
| 1   | 大学物理 | 3       | 李老师   |
| 1   | C 语言   | 3       | 李老师   |
| 2   | 数据结构 | 3       | 赵老师   |
| 3   | 大学语文 | 3       | 高老师   |
...
+-----+-----+-----+-----+
3 rows in set (0.00 sec)
```

### 13.6.7 使用 CONCAT 联合多列

在 SELECT 语句中使用 CONCAT 函数，可以联合几个字段，构成一个总的字符串。下面的示例中，把学生的信息合并在一起显示，如下所示。

```
mysql> SELECT CONCAT(sname,":",sage) FROM students;
+-----+
| CONCAT(sname,":",sage) |
+-----+
| 张三:21                 |
| 李四:22                 |
+-----+
```

联合字段的字段名为 CONCAT 函数形成的表达式，这样看上去非常不方便，常常需要使用 MySQL 中的 AS 关键字，对联合字段起一个别名，如下所示。

```
mysql> SELECT CONCAT(sname,":",sage) AS basic_info FROM students;
+-----+
| basic_info |
+-----+
| 张三:21    |
| 李四:22    |
+-----+
```

在这里，basic\_info 是给联合字段起的别名，可以给它加上双引号，也可以不加，不过在有的地方必须要加，例如别名含有空格或者标点符号。

### 13.6.8 使用 LIMIT 限定结果行数

利用 LIMIT 子句，可以在查询返回许多行时，只显示其中的几行。例如，下面查询数据结构课程的前 5 名：

```
mysql> SELECT sname FROM students,stu_cou,courses
-> WHERE students.sid=stu_cou.sid AND courses.cid=stu_cou.cid
-> AND courses.cname='数据结构'
-> ORDER BY score
-> LIMIT 5;
```



```

+-----+
| sname |
+-----+
| 李四  |
| 张三  |
...
+-----+
5 rows in set (0.11 sec)

```

LIMIT 也可以从查询结果中取出中间部分。为了做到这一点，必须指定两个参数。第一个参数为希望看到的第一个记录编号（第一个结果记录的编号为 0 而不是 1）。第二个参数为希望查询的记录个数。下面的查询类似于前面的查询，但显示从第 10 名到第 20 名的学生：

```

mysql> SELECT sname FROM students,stu_cou,courses
-> WHERE students.sid=stu_cou.sid AND courses.cid=stu_cou.cid
-> AND courses.cname='数据结构'
-> ORDER BY score
-> LIMIT 9 10;

```

### 13.6.9 使用函数和表达式

在前面的各节中，多数查询通过从表中检索已有的值产生输出结果。此外，MySQL 还允许用一个表达式来计算各列的值，作为输出结果。表达式可以简单也可以复杂，还可以包含一些函数。例如，下面的 SQL 语句计算张三的总成绩，并且取别名为 sum\_score：

```

mysql> select sum(score) as sum_score from students,stu_cou
-> where students.sid=stu_cou.sid and students.sname='张三';
+-----+
| sum_score |
+-----+
|      178 |
+-----+

```

而下面的 SQL 语句给使用更为复杂的表达式，计算张三的平均学分积，并且取别名为 avg\_score：

```

mysql> select sum(score)/sum(ccredit) as avg_score from students,stu_cou,courses
-> where students.sid=stu_cou.sid and students.sname='张三';
+-----+
| avg_score |
+-----+
|    29.67 |
+-----+

```

MySQL 提供了很多的函数可以使用，包括数学函数、比较函数、字符串处理函数，以及日期时间函数等等，此处不一一详述，在实际开发中使用到时，请参考 MySQL 函数手册。除函数之外，SQL 语句还可以使用算数运算符、字符串运算符，以及逻辑运算符来构成表达式。下面示例根据年龄得到出生年信息：

```

mysql> select *,year(now())-sage as sbirthday from students;
+-----+-----+-----+-----+-----+
| sid | sname | sgender | sage | sbirthday |
+-----+-----+-----+-----+-----+
| 1 | 张三 | 男 | 21 | 1985 |
| 2 | 李四 | 男 | 22 | 1984 |
+-----+-----+-----+-----+-----+

```

## 13.7 查询数据库、表信息

上两节介绍了如何增删改查数据，是直接对数据的管理，本节继续介绍如何获取数据库、数据表的信息。虽然这些操作不直接管理数据，但往往是存储和管理数据的基础。

MySQL 提供了几个获取数据库和表中信息的语句：

### 1 . SHOW

用来获取数据库和表的几个方面的信息，有如下用法：

SHOW DATABASES：列出服务器上的数据库

SHOW TABLES：列出当前数据库中的表

SHOW TABLES FROM db\_name：列出指定数据库中的表

SHOW COLUMNS FROM tbl\_name：显示指定表中列的信息

SHOW INDEX FROM tbl\_name：显示指定表中索引的信息

SHOW TABLE STATUS：显示缺省数据库中表的说明信息

SHOW TABLE STATUS FROM db\_name：显示指定数据库中表的说明信息

### 2 . DESCRIBE / EXPLAIN

DESCRIBE tbl\_name 和 EXPLAIN tbl\_name 语句与 SHOW COLUMNS FROM tbl\_name 功能相同。例如，下面是上面命令的一些示例：

```
mysql> use student_course;
Database changed
mysql> show tables;
+-----+
| Tables_in_student_course |
+-----+
| courses                  |
| stu_cou                  |
| students                  |
+-----+
3 rows in set (0.00 sec)

mysql> show columns from students;
+-----+-----+-----+-----+-----+-----+
| Field | Type      | Null | Key | Default | Extra |
+-----+-----+-----+-----+-----+-----+
| sid   | int(11)   |      |     |          |       |
| sname | varchar(50)|      |     |          |       |
| sgender | char(2)   | YES  |     |          |       |
| sage  | int(11)   | YES  |     |          |       |
+-----+-----+-----+-----+-----+-----+
```

### 3 . mysqlshow

mysqlshow 命令提供了某些与 SHOW 语句相同的信息，它允许从外壳程序中取得数据库和表的信息：

% mysqlshow 列出服务器上的数据库

% mysqlshow db\_name 列出指定数据库中的表

% mysqlshow db\_name tbl\_name 显示指定表中列的信息

% mysqlshow --keys db\_name tbl\_name 显示指定表中索引的信息

% mysqlshow --status db\_name 显示指定数据库中表的说明信息

#### 4 . mysqldump 实用程序

以 CREATE TABLE 语句的形式查看表的结构。

这些语句可以用来了解数据库的内容及表的结构 ,还可以作为使用 ALTER TABLE 的辅助手段 , 获取当前列的信息 , 计划出怎样对列进行修改。

## 13.8 MySQL 不支持的功能

前面讲述了使用 MySQL 来管理数据 ,体现了 MySQL 的强大功能 ,而本节则将介绍 MySQL 不支持的功能。MySQL 之所以忽略某些功能是因为它们有负面性能影响 ,而有的功能正在计划开发中 ,一旦找到既可以实现相应的功能 ,又不致于影响良好性能的方法 ,就会实现它们。

### 1 . 嵌套查询

嵌套查询是指一个 SELECT 语句嵌套在另一个 SELECT 语句内部 ,如下面示例 ,希望获取所有不及格的学生信息 :

```
SELECT* FROM students
WHERE sid IN (
    SELECT sid FROM stu_cou AND score<60
)
```

这时 ,可以通过分析需求 ,将嵌套查询改变为不含子查询的 SQL 语句来完成。

### 2 . 事务处理

事务处理是指作一组 SQL 语句为一个整体执行而不会被中断。提交/回退事务允许数条语句作为一个整体全部执行或全部不执行。即 ,如果事务处理中的任何一条语句失败 ,那么事务中所有语句都被撤消。

MySQL 自动进行单一 SQL 语句的同步。例如 ,两个客户机不能对相同的表进行同时写入。此外 ,可利用 LOCK TABLES 和 UNLOCK TABLES 对表加/解锁 ,把数条 SQL 语句组成一个整体 ,完成单条语句不能完成的功能。

MySQL 与事务处理有关的缺陷是 ,它不能自动对数条语句进行事务控制 ,如果这些语句中有某一条失败后 ,也不能对整个事务进行回退。

### 3 . 外键和引用完整性

外键定义一个表中的键与另一个表中的键相关 ,引用完整性对包含外键的表的数据进行一定的约束 ,以避免破坏数据的引用完整性。

例如 ,student\_course 数据库的 stu\_cou 表中包含学生编号 ( sid ) 列 ,用来将学分信息关联到 students 表中的学生。此时 ,score.student\_id 为一个外键 ,引用完整性对其约束 :他的值不能为 students 表中不存在的学生编号。此外 ,引用完整性还支持级联删除 :如果某个学生从 student 表中被删除后 ,该学生的任何学分记录将会自动地从 score 表中删除。

然而 ,除了在一定程度上能够在 CREATE TABLE 语句中分析 FOREIGN KEY 子句外 ,MySQL 不支持外键。MySQL 不强制让外键作为一种约束 ,也不提供级联删除功能。MySQL 不支持外键的原因主要是由于它对数据库的实现与维护有负作用。

要实现 DELETE 的级联效果 ,使用 MySQL 用户可以用应用程序来完成。例如 ,假如想要删除 sid=1 的学生 ,这也隐含要求需要删除该学生的选课记录 ,可以利用程序来做到这一点。而在支持级联删除的数据库 ( 如 SQLServer ) 中 ,只需要用如下的语句就可以删除 students 表的记录和相应的 stu\_cou 表的记录 :

```
delete from students where sid=1
```

```
delete from stu_cou where sid=1
```

#### 4. 存储过程和触发器

存储过程是事先编译好的 SQL 代码脚本，完成特定的数据操作功能。而触发器是一种特殊的存储过程，在对数据进行某个操作时，被激活并自动运行。例如，当需要对某一列进行计数统计时，使用某一列上的值存储这个统计值。当某个记录被删除时，就应该重新进行统计，使结果反映最新情况，这时便可用 Delete 类型的触发器，触发器的功能是统计，而 Delete 操作会触发执行该功能。

存储过程已列入了 MySQL 准备实现的计划。

#### 5. 视图

视图是一个逻辑概念，其功能像表但本身不是表，其本质为物理表的逻辑呈现，因此也称为虚表。MySQL 也准备实现视图功能。

## 本章小结

本章的内容很多，包括了使用 MySQL 管理数据的各个方面。

首先，本章介绍了与 MySQL 交互所使用的语言 SQL，它是使用关系数据库的基础。然后分别介绍了如何使用 MySQL 来对数据库和数据表进行管理，具体包括其创建、修改、删除操作。索引是优化查询的重要技术。

在第 5、6 节，详细介绍了在开发中使用最多的增删改查数据操作，这是真正意义上的数据管理。最后，介绍了如何获取数据库以及数据表的信息，以及 MySQL 所不支持的一些功能。

## 第 19 章 支持信息加密的用户验证

上一章增强了聊天室的功能，其中包括对用户注册信息的管理。细心的读者很容易发现，对于用户口令这个重要的数据，聊天室的处理非常不安全：

（1）未被加密过的密码在网络上传输时，很容易被窃取

（2）密码存储在数据库中，也应当经过加密以防止泄漏

就像利用 MySQL 的 `password()` 函数将 MySQL 用户口令以加密形式存储一样，Web 应用中的用户口令信息，以及其他重要的信息，也应当在被加密后传输或存储，在使用时，则需要重新解密。

### 19.1 加密技术简介

信息加密和解密的目的在于保护其不被非法使用或修改。其中，加密是将易懂的信息转换为不易懂信息，而解密则是将不易懂的信息转换为原来易懂信息。不易懂的信息称作密文，易懂的信息称作明文。

说明：如果读者对密码学的基本概念已经非常熟悉，就可以跳过去这一部分。

#### 19.1.1 加密的功能

加密可以保护数据不被查看和修改，并且可以在本不安全的网络上提供安全的通信方式。例如，可以使用加密算法对数据进行加密，在加密状态下传输数据，然后由接收方进行解密。如果第三方截获了加密的数据，也不知所云。

信息加密的功能可以通过一个简单的例子来说明。例如，有两个人地瓜和土豆需要在不安全的信道上通信，他们想要确偷听者无法理解他们之间的交流信息，为了做到这一点，需要保证：

（1）由于地瓜和土豆相距遥远，因此地瓜必须确保她从土豆处收到的信息没有在传输期间被别人修改。

（2）地瓜必须确定信息确实是发自土豆，而不是别人模仿土豆发出的。

信息加密可以达到目标，具体来说，信息加密可以达到以下功能：

保密性：帮助保护用户的标识或数据不被读取。

数据完整性：帮助保护数据不更改。

身份验证：确保数据发自特定的一方。

为了达到这些目的，通常需要使用加密算法和密钥的组合实现信息加密。

#### 19.1.2 常用加密方法

前面介绍了加密/解密的功能，下面分别简单介绍常用的密钥及加密原理。

##### 1. 私钥加密

私钥加密又称为对称加密，同一密钥既用于加密又用于解密。由于具有密钥的任意一方都可以使用该密钥解密数据，因此必须保护密钥不被别人得到。

## 2. 公钥加密

公钥加密算法也称为不对称算法，原因是需要用一个密钥加密数据，而用另一个密钥来解密数据。其中，加密数据所用的密钥可以对任何人公开，称为公钥；而解密数据所用的密钥必须保密，称为私钥。

公钥和私钥在数学上相关联；用公钥加密的数据只能用私钥解密，而用私钥签名的数据只能用公钥验证。

## 3. 数字签名

公钥算法还可用于构成数字签名。数字签名验证发送方的身份，并保护数据的完整性。例如，使用由地瓜生成的公钥，地瓜的数据的接收者可以通过将数字签名与地瓜的加密数据进行比较，来验证该数据是否由地瓜发送。

## 4. 散列

散列算法将任意长度的二进制值映射为固定长度的较小二进制值，小的二进制值称为散列值。散列值是一段数据唯一且极其紧凑的数值表示形式。散列一段明文后，哪怕只更改一个字母，生成的散列都将变化很大。已经证明，要找到散列为同一个值的两个不同的输入，在计算上是不可能的。

# 19.2 XOR 运算加密算法

上一节简单介绍了加密技术的基本功能及常用方法，本节将介绍一个易用简单的加密/解密算法：使用异或（XOR）运算。本算法原理简单，旨在使读者对信息的加密/解密有一个更加直观的印象。

## 19.2.1 XOR 算法原理

从加密的主要方法看，换位法过于简单，特别是对于数据量少的情况很容易由密文猜出明文，而替换法不失为一种行之有效的简易算法。

从各种替换法运算的特点看，异或运算最适合用于简易加解密运算，这种方法的原理在于：当一个数 A 和另一个数 B 进行异或运算会生成另一个数 C，如果再将 C 和 B 进行异或运算则 C 又会还原为 A。

相对于其他的简易加密算法，XOR 算法的优点包括：

- （1）算法简单，对于高级语言很容易能实现
- （2）速度快，可以在任何时候、任何地方使用
- （3）对任何字符都是有效的，不像有些简易加密算法，只对西文字符有效，对中文加密后再解密无法还原为原来的字符。

## 19.2.2 XOR 算法实现

上一部分介绍了如何使用 XOR 运算进行加密/解密的原理，本节将使用其加密用户的登录信息。根据上一小结介绍的 XOR 加密算法的原理，不难写出以下的加密解密函数。首先来看

加密算法：

```
1. <!--encryp_xor:简单使用 XOR 运算的加密函数----->
2. <?php
3. //加密函数
4. function myEncrypt($string, $key)
5. {
6.     for($i=0; $i<strlen($string); $i++)
7.     {
8.         for($j=0; $j<strlen($key); $j++)
9.         {
10.             $string[$i] = $string[$i]^$key[$j];
11.         }
12.     }
13.     return $string;
14. }
```

第 4 行定义了加密函数 myEncrypt()，输入参数 \$string 为明文，而 \$key 为密钥；输出为使用 \$key 作为密钥并使用 XOR 加密算法产生的密文。

第 6-12 的外层 for 循环对明文字符串的每一个字符进行循环，而内层的 for 循环（第 8-11 行）对明文的每一字符循环与密钥的每一位做异或运算。其原理已经在上一小结主中介绍，不再重述。

同样，与加密函数类似，可以写出下面的解密函数：

```
1. //解密函数
2. function myDecrypt($string, $key)
3. {
4.     for($i=0; $i<strlen($string); $i++)
5.     {
6.         for($j=0; $j<strlen($key); $j++)
7.         {
8.             $string[$i] = $key[$j]^$string[$i];
9.         }
10.    }
11.    return $string;
12. }
13. ?>
```

第 4 行定义了解密函数 myDecrypt()，输入参数 \$string 为密文，而 \$key 为密钥；输出为使用 \$key 作为密钥并使用 XOR 解密算法产生的明文。

下面，通过一个应用示例来进一步说明加密函数的功能：

```
1. //示例
2. $my_password="chair";
3. echo "my_password = $my_password<br>";
4. $my_key="1234567890";
5. $my_password_en=myEncrypt($my_password,$my_key);
6. echo "my_password_en = $my_password_en<br>";
7. $my_password_de=myDecrypt($my_password_en,$my_key);
8. echo "my_password_de = $my_password_de<br>";
```

第 3 行首先定义了一个明文 \$my\_password，然后在第 4 行定义密钥 \$my\_key。

第 5、6 行分别调用加密函数生成密文并输出；反过来，又在第 7、8 行将密文解密。

上面示例的运行结果如下：

```
my_password = chair
my_password_en = RYPXC
my_password_de = chair
```

### 19.2.3 用 XOR 算法实现身份验证

上两部分分别介绍了使用 XOR 运算进行信息加密/解密的原理和实现，下面，将使用这一方法来对用户的登录密码进行加密。本例中，为了保护用户的密码，系统想要达到的目的包括：

在用户注册时，用户需要添写用户密码表单；

除用户本人之外，其他任何人都无法获取其密码信息，包括系统设计者和数据库管理员；

系统能根据用户输入的密码验证用户的合法性。

为了达到以上目的，使用 XOR 算法时可以选择用户名作为明文，而密钥是用户自定义的密码，然后将加密后的用户名存储在数据库中。

另外，在用户登录的时候，有两种方式来验证合法用户：

（1）根据其提交的用户名（明文）和密码（密钥）信息重新加密，并使用加密后的信息与数据库中存储的密码信息进行比较，如果相等，则用户合法，否则，为非法用户。

（2）根据数据库中存储的密码信息（明文）和用户输入的密码（密钥）信息进行解密，并把解密后的信息与用户提交的用户名进行比较，如果相等，则用户合法，否则，为非法用户。

两种方式都可以实现第三个条件，本例，将采用第二种方式。本例的实现代码可在 18.4.1 节“用户登录”和 18.4.2 节“检查用户”的实现基础之上实现，其中“用户登录”页面无需变化，“检查用户”的实现参考如下：

```
1.  <?php
2.      session_start();                //装载 Session 库，一定要放在首行
3.      $user_name=$_POST["user_name"];
4.      session_register("user_name");  //注册$user_name 变量,注意没有$符号
5.
6.      require_once("sys_conf.inc");    //系统配置文件，包含数据库配置信息
7.      require_once("encrypty_xor.php"); //包含 xor 加密函数文件
8.
9.      //连接数据库
10.     $link_id=mysql_connect($DBHOST,$DBUSER,$DBPWD);
11.     mysql_select_db($DBNAME);        //选择数据库 my_chat
12.
13.     //查询是否存在登录用户信息
14.     $str="select name,password from user where name ='$user_name'";
15.     $result=mysql_query($str,$link_id); //执行查询
16.     @$rows=mysql_num_rows($result);    //取得查询结果的记录笔数
17.     $user_name=$_SESSION["user_name"];
18.     $password=$_POST["password"];
19.     $password_en=myEncrypt($user_name,$password); //加密用户信息
20.
21.     //对于老用户
22.     if($rows!=0)
23.     {
24.         list($name,$pwd)=mysql_fetch_row($result);
25.         $password_de=myDecrypt($pwd,$password); //解密用户信息
26.
27.         //如果密码输入正确
28.         if($user_name==$password_de)
29.         {
30.             $str="update user set is_online =1 where name ='$user_name' and
password='$password_en'";
31.             $result=mysql_query($str, $link_id); //执行查询
```



```

32.         require("main.php"); //转到聊天页面
33.     }
34.     //密码输入错误
35.     else
36.     {
37.         require("relogin.php");
38.     }
39. }
40. //对于新用户，将其信息写入数据库
41. else
42. {
43.     $str="insert          into          user          (name,password,is_online)
values('$user_name','$password_en',1)";
44.     $result=mysql_query($str, $link_id); //执行查询
45.     require("main.php"); //转到聊天页面
46. }
47. //关闭数据库
48. mysql_close($link_id);
49. ?>

```

第 7 行引入了加密函数文件 encrypy\_xor.php，包括上一小结介绍的两个函数。

第 19 行，使用用户提交的用户名和密码得到加密后的密码值，并且对于新用户，在第 44 行将这个加密后的值存储在数据库中。

另外，对于老用户，在第 24 获取数据库中用户名和加密后的密码信息，并在 25 行利用这两个值进行解密，然后在第 28 行通过比较解密后的值与用户提交的用户名信息来检查用户的合法性。

#### 19.2.4 自动生成密钥

上一部分介绍了如何使用 XOR 加密算法进行对用户信息的加密，其中，用户所输入的口令信息实际上成为了加密算法中的密钥，而用户名作为明文使用，虽然这能很好的完成功能，但是在逻辑上，这中方法似乎有些别扭。

本节，就将介绍一种自动生成密钥的技术，可以使用这个自动生成的密钥对用户提交的密码明文加密，使逻辑更加合理一些。

本例，假设我们生成的密钥为 512 位。代码如下：

```

1.  <!--keygen.php:自动生成密钥----->
2.  <?php
3.
4.      //自动生成长度为$len 的密钥
5.      function generate_key($len)
6.      {
7.          $lowerbound = 35 ;
8.          $upperbound = 96 ;
9.          $strMyKey = "";
10.
11.         for($i=1;$i<=$len;$i++)
12.         {
13.             $rnd=rand(0,100);          //产生随机数
14.             $k = (($upperbound - $lowerbound) + 1) * $rnd + $lowerbound;
15.             $strMyKey=$strMyKey.$k;

```

```

16.     }
17.     return $strMyKey;
18. }
19.
20. //将密钥写入文件$file_name
21. function write_key($key,$file_name)
22. {
23.     $filename="C:\key.txt";
24.     $key=generate_key($key,512);
25.
26.     //使用添加模式打开$filename，文件指针将会在文件的末尾
27.     if(!$handle=fopen($filename,'w'))
28.     {
29.         print"不能打开文件$filename";
30.         exit;
31.     }
32.
33.     //将$key 写入到我们打开的文件中。
34.     if(!fwrite($handle,$key))
35.     {
36.         print"不能写入到文件$filename";
37.         exit;
38.     }
39.     fclose($handle);
40. }
41.
42. //读取密钥文件中的密钥
43. function get_key($file_name)
44. {
45.     //打开文件
46.     $fp = fopen ($file_name, "r");
47.     $result="";
48.     //逐行读取
49.     while (!feof($fp))
50.     {
51.         $buffer = fgets($fp, 4096);
52.         $result=$result.$buffer;
53.     }
54.     return $result;
55. }
56.
57. /**
58.  $KeyLocation = "C:\key.txt";           //保存密钥的文件
59.  $key="123456";
60.  write_key($key,$KeyLocation);
61.  echo get_key($KeyLocation);
62.  /**/
63.  ?>

```

代码包括三个函数：

generate\_key(\$len)：自动生成长度为\$len 的密钥

write\_key(\$key,\$file\_name)：将密钥写入文件\$file\_name

get\_key(\$file\_name)：读取密钥文件\$file\_name 中的密钥值

在使用时，当用户第一次登录系统时，自动为其生成密钥值，对于这个密钥值，可以有两种方式来处理：

（1）将其存入数据库的某个字段中，这种方法的缺点是密钥在数据库中的安全性无法得到保证；

（2）将这个密钥保存在用户本地的文件中，这样就可以避免密钥被别人所获取，但这种方式的缺点是，当用户使用其他机器访问系统时，就无法登录。

本例中，将使用第二种方式。

具体地，上面代码第 11-18 行通过生成随机数的方式来不断生成密钥，并通过一个计算来增强其复杂性。其中的 lowerbound 和 upperbound 的数值其实就是你使用来加密的 ASCII 字符范围。下面是生成的一个密钥文件示例：

```
208123915925183361116049369344372701567721435181102718332639307390344373445407
524316475863232913993383189547474747394154915312639841226741894189965623523913
011164730113445201935692839710274127251577929493941487145611337531549110895367
593586318332391170941272701152344371709270125776235313540032267139933835677407
617384135696111239130732949469623520815987524358635491542913374933524334454251
400327015367133759324537171709152357391089524342514685239122673135531363151191
833412771743139654...
```

最后，需要把密钥保存在服务器上的一个安全的地方，然后就可以利用其和诸如 XOR 这样的加密算法来对用户信息进行加密/解密了。如何在上一部分介绍的 XOR 中使用这个密钥非常简单，不再详述。

## 19.3 用 crypt()实现用户身份验证

上一节介绍了一个简单的加密算法，读者可以以此为砖，引来类似的加密算法。另外，如果不想自己开发新的加密算法，还可以利用 PHP 提供的 crypt()函数来完成单向加密功能。

### 19.3.1 了解 crypt()

只要有一点使用非 Windows 平台经验的读者都可能对 crypt()相当熟悉，这一函数完成被称作单向加密的功能，它可以加密一些明码，但不能反过来将密码重新转换为原来的明码。

crypt()函数定义如下：

```
string crypt (string input_string [, string salt])
```

其中，input\_string 参数是需要加密的明文字符串，第二个可选的 salt 是一个位字符串，能够影响加密的暗码，进一步地排除被破解的可能性。缺省情况下，PHP 使用一个 2 个字符的 DES 干扰串，如果系统使用的是 MD5（参考下一节内容），PHP 则会使用一个 12 个字符的干扰串。可以通过执行下面的命令发现系统将要使用的干扰串的长度：

```
print "My system salt size is: ". CRYPT_SALT_LENGTH;
```

crypt()支持四种加密算法，表 19.1 显示了其支持的算法和相应的 salt 参数的长度：

表 19.1 crypt()支持四种加密算法

算法	Salt长度
CRYPT_STD_DES	2-character (Default)
CRYPT_EXT_DES	9-character
CRYPT_MD5	12-character beginning with \$1\$
CRYPT_BLOWFISH	16-character beginning with \$2\$

从表面上来看，crypt()的函数功能似乎没有什么用处，但该函数的确被广泛用来保证系统密

码的完整性。因为，单向加密的口令即使落入第三方人的手里，由于不能被还原为明文，也没有什么大用处。

### 19.3.2 用 crypt()实现用户身份验证

上一部分简单介绍了 crypt()函数的功能，下面利用其来实现用户的身份验证，其所要实现的目标同 19.2.3 所介绍的一致。

```
1.  <!--check_user_crypt.php：使用 crypt()函数验证用户----->
2.  <?php
3.      $user_name=$_POST["user_name"];
4.      require_once("sys_conf.inc");          //系统配置文件，包含数据库配置信息
5.
6.      //连接数据库
7.      $link_id=mysql_connect($DBHOST,$DBUSER,$DBPWD);
8.      mysql_select_db($DBNAME);              //选择数据库 my_chat
9.
10.     //查询是否存在登录用户信息
11.     $str="select name,password from user where name ='$user_name'";
12.     $result=mysql_query($str,$link_id);      //执行查询
13.     @$rows=mysql_num_rows($result);          //取得查询结果的记录笔数
14.     $user_name=$_SESSION["user_name"];
15.     $password=$_POST["password"];
16.     $salt = substr($password, 0, 2);
17.     $password_en=crypt($password,$salt);     //使用 crypt()对用户密码进行加密
18.
19.     //对于老用户
20.     if($rows!=0)
21.     {
22.         list($name,$pwd)=mysql_fetch_row($result);
23.
24.         //如果密码输入正确
25.         if($pwd==$password_en)
26.         {
27.             $str="update user set is_online =1 where name ='$user_name' and
password='$password_en'";
28.             $result=mysql_query($str, $link_id); //执行查询
29.             require("main.php");                //转到聊天页面
30.         }
31.         //密码输入错误
32.         else
33.         {
34.             require("relogin.php");
35.         }
36.     }
37.     //对于新用户，将其信息写入数据库
38.     else
39.     {
40.         {
41.             $str="insert          into          user          (name,password,is_online)
values('$user_name','$password_en',1)";
42.             $result=mysql_query($str, $link_id); //执行查询
```

```

43.         require("main.php");           //转到聊天页面
44.     }
45.     //关闭数据库
46.     mysql_close($link_id);
47. ?>

```

示例与上一节所介绍的使用 XOR 加密算法来保护用户信息非常类似，其核心部分在于第 16、17 行使用 `crypt()` 函数获取加密后的密码，而通过在第 25 行比较数据库中的密码和加密后的密码是否相等来检查用户是否合法。

下面，通过一个实例来看一下加密后的密码会变成什么样子。

例如，用户名为 rock，密码为 123456，则加密后的密码为：

```
12tir.zlbWQ3c
```

上面就实现了一个验证用户的简单身份验证系统。在使用 `crypt()` 保护重要的机密信息时，需要注意的是，在缺省状态下使用的 `crypt()` 并不是最安全的，只能用在对安全性要求较低的系统中，如果需要较高的安全性能，就需要我在本篇文章的后面介绍的算法。

## 19.4 MD5 散列加密算法

上一节介绍了使用 `crypt()` 来对信息进行加密，另外，还可以使用 PHP 内置的 MD5 算法来实现同样的功能。本节，就将对 MD5 算法进行详细的介绍。

### 19.4.1 了解 MD5

MD5 的全称是 Message-Digest Algorithm 5，作用是让大容量信息在用数字签名软件签署私人密匙前被“压缩”成一种保密的格式，即使用一个散列函数，把一个任意长度的字节串转换成一定长的大整数。

散列函数可以将一个可变长度的信息变换为固定长度输出，也被称作“信息摘要”。这十分有用，因为固定长度的字符串可用来检查文件的完整性和验证数字签名，以及用户身份验证等。

下面来看一个最简单的使用 MD5 散列一个字符串的示例：

```

<!--md5_test.php:使用 md5()混编字符串----->
<?php
    $msg = "这是一个明文字符串";
    echo "msg: $msg". "<br>";
    $enc_msg = md5($msg);
    echo "hash: $enc_msg ". "<br>";
?>

```

上面代码调用了 PHP 的内置 `md5()` 函数，将一个明文字符串散列为一个定长的密文。代码的输出结果如：

```

msg: 这是一个明文字符串
hash: 06f8a3f0a1190ef25cc4556af7a7cdbe

```

需要注意的是，结果串长度为 32 个字符，无论明文是怎样的，这个长度总是一定的。

## 19.4.2 使用 MD5 实现用户身份验证

上一部分简单介绍了 MD5 算法的基本原理和其强大的加密功能，并用一个简单的示例来说明如何使用 PHP 内置的 md5() 函数，下面，将进一步介绍使用其实现用户的身份验证的问题。PHP 内置的 md5() 散列函数将把一个可变长度的信息转换为 128 位(32 个字符)的信息文摘。如前所述，散列的特点是不能通过分析散列信息得到原来的明码，因为散列后的结果与原来的明码内容没有依赖关系。即便只改变明文的字符串中的一个字符，也将使得 MD5 散列算法计算出二个截然不同的结果。

```
1.  <?php
2.      $user_name=$_POST["user_name"];
3.
4.      require_once("sys_conf.inc");           //系统配置文件，包含数据库配置信息
5.
6.      //连接数据库
7.      $link_id=mysql_connect($DBHOST,$DBUSER,$DBPWD);
8.      mysql_select_db($DBNAME);               //选择数据库 my_chat
9.
10.     //查询是否存在登录用户信息
11.     $str="select name,password from user where name ='$user_name'";
12.     $result=mysql_query($str,$link_id);      //执行查询
13.     @$rows=mysql_num_rows($result);          //取得查询结果的记录笔数
14.     $user_name=$_SESSION["user_name"];
15.     $password=$_POST["password"];
16.     $password_en=md5($password);             //使用 md5t()对用户密码进行加密
17.
18.     //对于老用户
19.     if($rows!=0)
20.     {
21.         list($name,$pwd)=mysql_fetch_row($result);
22.
23.         //如果密码输入正确
24.         if($pwd==$password_en)
25.             //...
26.         //密码输入错误
27.         else
28.             //...
29.     }
30.
31.     //对于新用户，将其信息写入数据库
32.     else
33.     {
34.         $str="insert into user
35.             (name,password,is_online) values('$user_name','$password_en',1)";
36.         $result=mysql_query($str, $link_id); //执行查询
37.         require("main.php");                 //转到聊天页面
38.     }
39.     //关闭数据库
40.     mysql_close($link_id);
41.     ?>
```

示例与第 3 节使用 crypt() 函数进行用户验证非常相似，其核心在于第 16 行对 md5() 算法的

调用。为避免重复，上述代码在第 25、28 行做了省略，其详细内容请参考 19.3.2 相关内容。

### 19.4.3 还原明文

前面介绍的 crypt() 和 md5() 在验证用户身份上非常好用，但二者在功能上都受到一定的限制，只能实现单向加密，而不能通过密文重新解密得到明文。下面介绍二个非常有用的被称作 Mcrypt 和 Mhash 的 PHP 扩展模块，将大大拓展用户在加密方面的选择。

Mcrypt 是一个功能强大的加密算法扩展库，它包括有 22 种加密算法。不过，在标准的 PHP 软件包中不包括 Mcrypt，因此需要下载它，可以从 <http://mcrypt.sourceforge.net/> 下载。下载后，对于 LINUX 系统，可以按照下面的方法进行编译，并把它扩充在 PHP 中：

```
gunzipmcrypt-x.x.x.tar.gz
tar -xvfmcrypt-x.x.x.tar
./configure --disable-posix-threads
make
make install
cd to your PHP directory.
./configure -with-mcrypt=[dir] [--other-configuration-directives]
make
make install
```

根据读者要求和 PHP 的安装配置环境不同，上面的过程可能需要作适当的修改。

Mcrypt 提供了 35 种加密函数，其优点不仅仅在于提供的加密算法较多，还在于它可以对数据进行加/解密处理。

下面的代码显示了如何使用 Mcrypt 扩展库，首先是对数据进行加密，然后在浏览器上显示加密后的数据，最后将加密后的数据还原为原来的字符串，将它显示在浏览器上：

```
1.  <!--Mycrypt.php 使用 Mcrypt 对数据进行加、解密----->
2.  <?php
3.  $string = "这是一个明文串 " ;           //明文串
4.  $key = "123456789 " ;                   //密钥
5.  $cipher_alg = MCRYPT_RIJNDAEL_128; // 选择加密算法
6.
7.  //生成增强安全性初始向量
8.  $iv = mcrypt_create_iv(  mcrypt_get_iv_size($cipher_alg,
                             MCRYPT_MODE_ECB), MCRYPT_RAND);
9.  print "Original string: $string <p>";      //输出明文

10. //加密，并输出密文
11. $encrypted_string = mcrypt_encrypt($cipher_alg, $key, $string, MCRYPT_MODE_CBC, $iv);
    //加密
12. Echo "Encrypted string: ".bin2hex($encrypted_string)."<p>"; // 输出密文
13.
14. //重新解密，得到明文
15. $decrypted_string =mcrypt_decrypt( $cipher_alg, $key,
                                       $encrypted_string, MCRYPT_MODE_CBC, $iv);
16. print "Decrypted string: $decrypted_string"; // 输出明文
17. ?>
```

第 11、15 行调用了加密和解密函数 mcrypt\_encrypt() 和 mcrypt\_decrypt()，其功能分别为把明文加密为密文，以及反过来由密文得到明文。

Mcrypt 提供了几种加密方式，本示例使用了“电报密码本”模式。每种加密方式都有可以

影响密码安全的特定字符 ,因此每种模式都需要了解。对于没有接触过密码系统的读者来说 ,可能需要首先了解一下对 `mcrypt_create_iv()`函数 ,需要利用其创建的初始化向量( hence, iv ),这一向量可使每条信息彼此独立。

执行上面的脚本将会产生下面的输出类似于 :

```
Original string: 这是一个明文
Encrypted string:
02a7c58b1ebd22a96554879694b091e60411cc4dea8652bb807234fa06bbfb20e71ecf525f29df58e
28f3d9bf541f7ebcecf62b89fde4d8e7ba1e6cc9ea24850478c11742f5cfa1d23fe22fe8 bfbab5e
Decrypted string: 这是一个明文
```

## 本章小结

不止于本例的应用 ,数据加密技术越来越重要。在本章中 ,将介绍常用的加密/解密技术 ,以及利用 PHP 的实现 ,并以加密用户口令为例进行说明。通过本章示例 ,读者可以增加所开发的 Web 系统的数据安全性。

本章介绍了如何对信息进行加密 ,主要包含三个部分 :首先介绍了一个简单的利用 XOR 操作实现的加密算法 ,并利用这个算法对用户信息进行加密 ,实现验证用户的功能。然后分别介绍了如何使用 PHP 内置的 `crypt()`和 `md5()`算法来实现数据的加密。



# 第 1 篇 PHP 开发篇

## 第 1 章 初识 PHP

- 1.1 初识 PHP
  - 1.1.1 走近 PHP
  - 1.1.2 搭建开发环境
  - 1.1.3 HelloWorld
- 1.2 第一个 PHP 程序：我的书房
  - 1.2.1 我的书房之欢迎光临
  - 1.2.2 我的书房之显示时间
  - 1.2.3 我的书房之临别赠言
- 1.3 在 HTML 中嵌入 PHP
  - 1.3.1 把 PHP 嵌入 HTML
  - 1.3.2 PHP 语句结束符
  - 1.3.3 注释程序
  - 1.3.4 引用文件
  - 1.3.5 PHP 与 C、Java
- 1.4 ASP、PHP、JSP 之比较
- 1.5 如何使用本书
- 本章小结

## 第 2 章 PHP 程序设计基础

- 2.1 数据类型
  - 2.1.1 理解数据类型
  - 2.1.2 字符串
  - 2.1.3 数字
  - 2.1.4 常用函数
- 2.2 变量
  - 2.2.1 理解变量
  - 2.2.2 声明变量
  - 2.2.3 变量赋值
  - 2.2.4 变量替换
  - 2.2.5 确定变量类型
  - 2.2.6 变量的作用域
  - 2.2.7 静态变量
  - 2.2.8 使变量名可变
- 2.2 常量
  - 2.2.1 定义和使用常量
  - 2.2.2 PHP 中的预定义常量
- 2.3 运算符
  - 2.3.1 算数运算符
  - 2.3.2 字符串运算符

- 2.3.3 赋值运算符
- 2.3.4 逻辑运算符
- 2.3.5 位运算符
- 2.3.6 其他运算符
- 2.3.7 运算符的优先级
- 2.4 程序结构
  - 2.4.1 使用 if 语句实现分支
  - 2.4.2 使用 switch 语句实现分支
  - 2.4.3 使用 while 语句实现循环
  - 2.4.4 使用 for 语句实现循环
  - 2.4.5 使用 break/continue 控制循环
- 2.5 函数
  - 2.5.1 理解函数的本质
  - 2.5.2 定义和调用函数
  - 2.5.3 在函数间传递参数
  - 2.5.4 使函数名可变
  - 2.5.5 使用递归
- 2.6 综合示例
- 本章小结

## 第3章 字符串和数组操作

- 3.1 操作字符串
  - 3.1.1 去除空格和其他特殊符号
  - 3.1.2 加入和去除反斜杠
  - 3.1.3 生成 HTML 元素
  - 3.1.4 分解字符串
  - 3.1.5 格式化字符串
  - 3.1.6 获取和替换子串
  - 3.1.7 定位字符
  - 3.1.8 求串长度
  - 3.1.9 获取 ASCII 编码
  - 3.1.10 比较字符串
  - 3.1.11 大小写转换
  - 3.1.12 小结
- 3.2 正则表达式
  - 3.2.1 理解正则表达式
  - 3.2.2 使用正则表达式
  - 3.2.3 构造正则表达式
  - 3.2.4 示例 1：验证 URL
  - 3.2.5 示例 2：验证电话号码
- 3.3 规则数组
  - 3.3.1 理解数组
  - 3.2.1 定义一个数组
  - 3.1.2 遍历数组元素

- 3.1.3 获取当前元素
- 3.1.4 改变数组大小
- 3.1.5 数组求交
- 3.1.6 合并两个数组
- 3.1.7 反转一个数组
- 3.1.8 获取多个元素
- 3.1.9 排序数组元素
- 3.1.10 综合示例
- 3.4 相联数组
  - 3.4.1 理解相联数组
  - 3.4.2 增加数组元素
  - 3.4.2 删除数组元素
  - 3.4.4 检测键是否存在
  - 3.4.5 检测值是否存在
  - 3.4.6 小结

本章小结

## 第4章 PHP的面向对象技术

- 4.1 面向对象技术简介
  - 4.1.1 理解面向对象思想
  - 4.1.2 理解类和对象
  - 4.1.3 用继承来重用代码
  - 4.1.4 用多态来统一对外
  - 4.1.5 是否选择面向对象
- 4.2 在PHP中使用类
  - 4.2.1 创建类和对象
  - 4.2.2 使用构造函数
  - 4.2.3 继承已有的类
  - 4.2.4 重载新的方法
  - 4.3.5 访问父类中的方法
- 4.3 PHP5的面向对象新特征
  - 4.3.1 控制访问权限
  - 4.3.2 静态属性和方法
  - 4.3.3 使用\_\_construct()和\_\_destruct()
  - 4.3.4 使用\_\_clone()克隆对象
  - 4.3.5 使用抽象方法和抽象类
  - 4.3.6 使用\_\_call()处理调用错误
  - 4.3.7 使用\_\_autoload()自动加载类
  - 4.3.8 把对象串行化

本章小结

## 第5章 PHP的MySQL数据库编程

- 5.1 WEB系统的体系结构
- 5.2 连接和关闭数据库

- 5.2.1 连接数据库
- 5.1.2 关闭数据库连接
- 5.3 数据库级操作
  - 5.3.1 创建、选择、删除数据库
  - 5.3.2 示例：《我的书房》之库级操作
- 5.4 增删改数据
  - 5.4.1 提交 SQL 语句
  - 5.4.2 示例：《我的书房》之增删改图书
- 5.5 查询数据
  - 5.5.1 使用 `mysql_result()`
  - 5.5.2 使用 `mysql_fetch_row()`
  - 5.5.3 使用 `mysql_fetch_array`
  - 5.5.4 使用 `mysql_fetch_object()`
  - 5.5.5 综合比较
- 5.6 获取数据库信息
  - 5.6.1 获取数据库基本信息
  - 5.6.2 获取数据表属性信息
  - 5.6.3 获取查询结果数目
- 5.7 其他常用操作
  - 5.7.1 处理错误信息
  - 5.7.2 释放内存
  - 5.7.3 切换用户
  - 5.7.4 获取插入记录自动标识
- 5.8 综合示例
  - 5.8.1 系统概述
  - 5.8.2 设置环境变量
  - 5.8.3 数据库类
  - 5.8.4 图书类
  - 5.8.5 主页面
  - 5.8.6 添加记录
  - 5.8.5 修改记录
  - 5.8.6 删除记录
  - 5.8.7 显示查询记录

本章小结

## 第6章 PHP 的文件系统操作

- 6.1 访问文件
  - 6.1.1 打开和关闭文件
  - 6.1.2 读取文件内容
  - 6.1.3 写入文件内容
  - 6.1.4 检测文件是否存在
  - 6.1.5 检测文件访问权限
  - 6.1.6 将文件内容赋值给数组
  - 6.1.7 复制文件

- 6.2 操作目录
  - 6.2.1 打开和关闭目录
  - 6.2.2 读取目录
  - 6.2.3 改变当前目录
  - 6.2.4 获得脚本文件目录
  - 6.2.5 使用目录对象
- 6.3 综合示例
  - 6.3.1 我的文件管理器
  - 6.3.2 我的文件查看器
- 本章小结

## **第 7 章 构建 PHP 动态网页**

- 7.1 与表单 (FORM) 交互
  - 7.1.1 认识表单
  - 7.1.2 获取 FORM 信息
  - 7.1.3 示例
- 7.2 管理 SESSION
  - 7.2.1 认识 Session
  - 7.2.2 注册 Session 变量
  - 7.2.3 使用 Session 变量
  - 7.2.4 注销 Session 变量
  - 7.2.5 示例：验证用户身份
- 7.3 页面跳转
  - 7.3.1 使用 header() 函数
  - 7.3.2 使用文件操作函数
- 7.4 HTML 模板
- 7.5 管理图形图像
  - 7.5.1 PHP 支持的图像类型
  - 7.5.2 管理画布
  - 7.5.3 获得图像大小
  - 7.5.4 向图像中加入文字
  - 7.5.5 画点
  - 7.5.6 画几何图形
  - 7.5.7 填充几何图形
- 7.6 使用时间和日期函数
  - 7.6.1 获取日期和时间
  - 7.6.2 格式化输出日期和时间
- 7.7 处理 URL 字符串
  - 7.7.1 分解 URL
  - 7.7.2 编码和解码 URL
- 本章小结

## **第 8 章 PHP 的网络编程技术**

- 8.1 网络通信

- 8.1.1 使用 POST 上传文件
  - 8.1.2 HTTP 认证
  - 8.1.3 Socket 通信
- 8.2 电子邮件
  - 8.2.1 打开一个邮箱
  - 8.2.2 发送电子邮件
  - 8.2.3 获取附件
  - 8.2.4 综合示例
- 8.3 使用 XML
  - 8.3.1 理解 XML
  - 8.3.2 利用数组生成 XML
  - 8.3.3 使用 DOM 创建 XML 文档
  - 8.3.4 使用 DOM 查询 XML 数据
  - 8.3.5 使用 DOM 删除 XML 数据
  - 8.3.6 使用 DOM 修改 XML 数据
  - 8.3.7 综合示例

本章小结

## 第 9 章 PHP 程序调试技术

- 9.1 基本调试策略
  - 9.1.1 使用错误报告
  - 9.1.2 使用 die 和 print 语句
- 9.2 使用 ECLIPSE 调试程序
  - 9.2.1 了解 PHPEclipse
  - 9.2.2 使用 PHPEclipse 调试语法错误
  - 9.2.3 使用 PHPEclipse 调试逻辑错误

本章小结

## 第 2 篇 MySQL 数据库管理篇

### 第 10 章 初识 MYSQL

- 10.1 关系数据库基础
  - 10.1.1 了解关系数据库
  - 10.1.2 一个示例
  - 10.1.3 数据库的功能
- 10.2 MySQL 概述
  - 10.2.1 了解 MySQL
  - 10.2.2 使用 MySQL
- 10.3 创建第一个 MYSQL 数据库
  - 10.3.1 连接 MySQL 服务器
  - 10.3.2 建立 student-course 数据库
  - 10.3.2 建立表
  - 10.3.3 添加数据

- 10.3.4 查询数据
- 10.3.5 修改数据
- 10.3.6 删除数据
- 10.3.7 数据在哪儿
- 10.4 MySQL 的数据类型
  - 10.4.1 数值型
  - 10.4.2 字符串型
  - 10.4.3 日期和时间类型
  - 10.4.4 NULL 值

本章小结

## 第 11 章 数据库设计理论

- 11.1 数据库设计概述
  - 11.1.1 数据库设计的定义
  - 11.1.2 数据库设计的目标
  - 11.1.3 数据库的体系结构
  - 11.1.4 数据库设计的步骤
- 11.2 需求分析
  - 11.2.1 需求分析的目标
  - 11.2.2 收集需求信息
  - 11.2.3 分析整理需求
  - 11.2.4 评审
- 11.3 概念设计
  - 11.3.1 理解 E-R 图数据模型
  - 11.3.2 用 E-R 图为现实建模
  - 11.3.3 示例
- 11.4 逻辑设计
  - 11.4.1 逻辑设计步骤
  - 11.4.2 理解关系的概念
  - 11.4.3 约束一个关系
  - 11.4.5 关系中的异常
  - 11.4.6 规范化关系
- 11.5 设计说明书示例

本章小结

## 第 12 章 管理 MySQL

- 12.1 概述
  - 12.1 MySQL 的构成
  - 12.2 管理概述
  - 12.3 保障数据安全
- 12.2 使用客户机
  - 12.2.1 使用 mysql 客户机
  - 12.2.2 使用 mysqladmin 客户机
  - 12.2.3 使用 mysqldump 客户机

- 12.3 MySQL 服务器的启动和关闭
  - 12.3.1 启动服务器
  - 12.3.2 连接服务器
  - 12.3.3 关闭服务器
- 12.4 用户和权限管理
  - 12.4.1 创建新用户和授权
  - 11.4.2 控制权限传递
  - 11.4.2 取消权限和删除用户
  - 11.4.3 了解 MySQL 的用户管理表
- 12.5 日志管理
  - 12.5.1 生成日志文件
  - 12.5.2 循环利用日志文件
- 12.6 备份数据库
  - 12.6.1 使用 mysqldump 备份
  - 12.6.2 使用直接拷贝数据文件方式
  - 12.6.3 两种方式的比较
- 12.7 恢复数据库
  - 12.7.1 恢复整个数据库
  - 11.7.2 恢复单个的表
- 本章小结

## 第 13 章 使用 MYSQL 管理数据

- 13.1 SQL 基础
  - 13.1.1 SQL 简介
  - 13.1.2 了解 SQL 语句
  - 13.1.3 在 SQL 中加注释
  - 13.1.4 MySQL 中的 SQL 特征
- 13.2 创建、删除和选择数据库
  - 13.2.1 创建数据库(CREATE DATABASE)
  - 13.2.2 删除数据库(DROP DATABASE)
  - 13.2.3 选择数据库(USE)
- 13.3 创建、修改、删除数据表
  - 13.3.1 创建表(CREATE TABLE)
  - 13.3.2 创建不存在的表(IF NOT EXISTS)
  - 13.3.3 创建临时表(TEMPORARY)
  - 13.3.4 利用 SELECT 的结果创建表
  - 13.3.5 修改表(ALTER TABLE)
  - 13.3.6 删除表(DROP TABLE)
- 13.4 创建和删除索引
  - 13.4.1 使用索引优化查询
  - 13.4.2 创建索引
  - 13.4.3 删除索引
- 13.5 增删改数据
  - 13.5.1 使用 INSERT 增加记录



- 13.5.2 使用 LOAD DATA 批量增加记录
- 13.5.3 使用 mysqlimport 批量增加记录
- 13.5.4 修改记录 (UPDATE)
- 13.5.5 删除记录 (DELETE)

### 13.6 查询数据

- 13.6.1 选择列
- 13.6.2 选择数据表
- 13.6.3 使用 WHERE 选择记录
- 13.6.4 使用 GROUP BY 对结果分组
- 13.6.5 使用 DISTINCT 对结果去重
- 13.6.6 使用 ORDER BY 对结果排序
- 13.6.7 使用 CONCAT 联合多列
- 13.6.8 使用 LIMIT 限定结果行数
- 13.6.9 使用函数和表达式

### 13.7 查询数据库、表信息

### 13.8 MySQL 不支持的功能

本章小结

## 第 14 章 查询优化

### 14.1 索引技术

- 14.1.1 了解索引机制
- 14.1.2 选择索引列
- 14.1.3 索引的缺点
- 14.1.4 分析索引效率

### 14.2 选择列类型

- 14.2.1 使用较短定长列
- 14.2.2 使用 NOT NULL 和 ENUM
- 14.2.3 使用 OPTIMIZE TABLE
- 14.2.4 避免检索较大的 BLOB 或 TEXT 值
- 14.2.5 使用 PROCEDURE ANALYSE()

### 14.3 优化 SQL 语句

- 14.3.1 避免对数据的顺序存取
- 14.3.2 避免通配符
- 14.3.3 比较两个列
- 14.3.4 使索引列独立
- 14.3.5 使用临时表
- 14.3.6 综合示例

### 14.4 系统优化

- 14.4.1 增加服务器缓存
- 14.4.2 使用查询缓存
- 14.4.3 调度与锁定数据
- 14.4.4 调整硬件

本章小结

## 第 15 章 MySQL 数据安全性

- 15.1 数据文件安全性
  - 15.1.1 保护哪些文件
  - 15.1.2 保护 Windows 文件
  - 15.1.3 保护 UNIX 文件
- 15.2 网络访问安全性
  - 15.2.1 通过授权表确认访客
  - 15.2.2 检查访问用户合法性
  - 15.2.3 控制库级访问
  - 15.2.4 控制表级访问
  - 15.2.5 控制列级访问
  - 15.2.6 使用授权表示例
- 15.3 数据库维护
  - 15.3.1 检查和维护数据库表
  - 15.3.2 使用 myisamchk 和 isamchk
  - 15.3.3 检查表
  - 15.3.4 修复表
  - 15.3.5 快速运行 myisamchk 和 isamchk
- 15.4 防范于未然
  - 15.4.1 建立预防性维护时间表
  - 15.4.2 在 LINUX 下定期检查表
  - 15.4.3 在 Windows 下定期检查表
- 本章小结

## 第 16 章 图形化管理 MYSQL

- 16.1 使用 PHPMYADMIN
  - 16.1.1 LINUX 系统的安装配置
  - 16.1.2 Windows 系统的安装配置
  - 16.1.3 库级操作
  - 16.1.4 表级操作
  - 16.1.5 增删改数据
  - 16.1.6 查询数据
  - 16.1.6 导出数据
- 16.2 其他图形管理工具
  - 16.2.1 MySQL Control Center
  - 16.2.2 MySQLGUI
  - 16.2.3 MySQL Administrator+MySQL Query Browser
- 本章小结

## 第 3 篇 典型模块设计篇

### 第 17 章 简易聊天室

- 17.1 系统目标
- 17.2 关键技术

- 17.2.1 自动刷新页面
- 17.2.2 传递用户名
- 17.2.3 显示最新发言
- 17.3 数据库设计
  - 17.3.1 设计数据库
  - 17.3.2 实现数据库
- 17.4 系统实现
  - 17.4.1 参数配置文件
  - 17.4.2 登录页面
  - 17.4.3 聊天室主页面
  - 17.4.4 显示发言页面
  - 17.4.5 发言页面
- 本章小结

## 第 18 章 支持用户管理的聊天室

- 18.1 系统目标
- 18.2 关键技术
  - 18.2.1 自动注册
  - 18.2.2 检查表单
  - 18.2.3 选择字体颜色
  - 18.2.4 显示字体颜色
  - 18.2.5 记录用户状态
- 18.3 数据库设计
  - 18.3.1 设计数据库
  - 18.3.2 实现数据库
- 18.4 系统实现
  - 18.4.1 用户登录
  - 18.4.2 检查用户
  - 18.4.3 重新登录
  - 18.4.4 聊天室主页面
  - 18.4.5 用户发言
  - 18.4.6 显示发言
  - 18.4.7 显示在线用户
  - 18.4.8 离开系统
- 本章小结

## 第 19 章 支持信息加密的用户验证

- 19.1 加密技术简介
  - 19.1.1 加密的功能
  - 19.1.2 常用加密方法
- 19.2 XOR 运算加密算法
  - 19.2.1 XOR 算法原理
  - 19.2.2 XOR 算法实现
  - 19.2.3 用 XOR 算法实现身份验证

- 19.2.4 自动生成密钥
- 19.3 用 CRYPT() 实现用户身份验证
  - 19.3.1 了解 crypt()
  - 19.3.2 用 crypt() 实现用户身份验证
- 19.4 MD5 散列加密算法
  - 19.4.1 了解 MD5
  - 19.4.2 使用 MD5 实现用户身份验证
  - 19.4.3 还原明文
- 本章小结

## 第 20 章 支持分页显示的拍卖行

- 20.1 系统目标
- 20.2 关键技术
  - 20.2.1 分页显示
  - 20.2.2 页面导航
  - 20.2.3 上传图片
  - 20.2.4 显示图片
  - 20.2.5 用户竞标
- 20.3 数据库设计
  - 20.3.1 设计数据库
  - 20.3.2 实现数据库
- 20.4 系统实现
  - 20.4.1 系统菜单
  - 20.4.2 配置文件
  - 20.4.3 分页显示类
  - 20.4.4 用户登录
  - 20.4.5 检查用户
  - 20.4.6 添加商品
  - 20.4.7 上传图片
  - 20.4.8 浏览商品
  - 20.4.9 显示商品详单
  - 20.4.10 出价竞标
- 本章小结

## 第 21 章 支持站内搜索的留言本

- 21.1 系统目标
- 21.2 关键技术
  - 21.2.1 组织留言内容
  - 21.2.2 分页显示留言信息
  - 21.2.3 传递留言 ID
  - 21.2.4 留言回复
  - 21.2.5 全文搜索留言信息
  - 21.2.6 独立搜索模块
- 21.3 数据库设计

- 21.3.1 设计数据库
- 21.3.2 实现数据库
- 21.4 系统实现
  - 21.4.1 添加留言页面
  - 21.4.2 站内搜索单元
  - 21.4.3 显示查询结果
  - 21.4.4 修改留言页面
  - 21.4.5 回复留言模块
  - 21.4.6 查看留言回复
  - 21.4.7 删除留言模块
  - 21.4.8 系统扩展
- 本章小结

## 第 4 篇 综合案例篇

### 第 22 章 开发大型项目的策略

- 22.1 开发规范标准化
  - 22.1.1 规范化命名
  - 22.1.2 规范化大括号使用
  - 22.1.3 规范化小括号使用
  - 22.1.4 规范化注释
  - 22.1.5 管理文档
  - 22.1.6 实施标准
- 22.2 WEB 系统的逻辑结构
  - 22.2.1 大型 Web 系统的结构
  - 22.2.2 API 化服务器端代码
  - 22.2.3 使用模板
- 本章小结

### 第 23 章 电子商务系统

- 23.1 需求分析
  - 23.1.1 电子商务简介
  - 23.1.2 用户行为分析
  - 23.1.3 系统目标
- 23.2 系统预览
  - 23.2.1 浏览、查询图书
  - 23.2.2 购买心仪的图书
  - 23.2.3 注册、登录系统
  - 23.2.4 查看我的购物篮
  - 23.2.5 生成我的购物订单
- 23.3 系统架构
  - 23.3.1 总体设计
  - 23.3.2 系统体系结构

- 23.4 数据库设计
- 23.5 数据访问层
  - 23.5.1 数据库配置文件
  - 23.5.1 DataBase 的属性
  - 23.5.2 实现 ExecuteSql()方法
  - 23.5.3 实现 Query ()方法
- 23.6 业务逻辑层
  - 23.6.1 图书类 Book
  - 23.6.2 购物车类 Cart
  - 23.6.3 会员类 User
  - 23.6.4 订单类 Order
  - 23.6.5 订单详细信息类 OrderDetail
- 23.7 页面显示层
  - 23.7.1 分页显示类
  - 23.7.2 图书浏览查询
  - 23.7.3 购物车管理
  - 23.7.4 会员管理
  - 23.7.5 订单管理
- 本章小结

## 第 24 章 办公自动化系统

- 24.1 需求分析
  - 24.1.1 办公自动化简介
  - 24.1.2 用户行为分析
  - 24.1.3 系统目标
- 24.2 系统预览
  - 24.2.1 进入办公平台
  - 24.2.2 撰写、上报公文
  - 24.2.3 审批公文
  - 24.2.4 发布、查看公告
  - 24.2.6 查看系统日志
- 24.3 系统架构
  - 24.3.1 总体设计
  - 24.3.2 系统体系结构
- 24.4 数据库设计
- 24.5 数据访问层
- 24.6 业务逻辑层
  - 24.6.1 用户类 User
  - 24.6.2 普通员工用户类 UserEmployee
  - 24.6.3 管理者用户类 UserManager
  - 24.6.4 角色类 Role
  - 24.6.5 公文类 File
  - 24.6.6 公文状态类 Status
  - 24.6.7 公文类属类 Category

24.6.8 公共消息类 PublicMessage

24.6.9 日志类 Log

24.7 页面显示层

24.7.1 分页显示类

24.7.2 系统首页

24.7.3 用户管理

24.7.3 个人办公

24.7.4 公告管理

24.7.5 日志管理

24.7.6 退出系统

本章小结

精通 PHP+MySQL 应用开发（附光盘）

【作者】王石，杨英娜 编著

【出版社】人民邮电出版社

【ISBN】7115149712

【出版日期】2006-7-1



章节精彩阅读：

<http://book.csdn.net/bookfiles/95/index.html>

购书网址：

<http://www.dearbook.com.cn/book/110808>

<http://www.china-pub.com/computers/common/info.asp?id=31354>

## 特点

140 个示例代码，全面展示 PHP 的各项技术

采用目标驱动，讲解了 MySQL 的各种用法

8 个典型案例，让您进入高手的行列

代码规范，注释详尽，方便阅读

作者经验的总结，代码可以直接应用于开发实际