## 索引数组赋值

索引数组赋值有三种方式:

第一种：用数组变量的名字后面跟一个中括号的方式赋值，当然，索引数组中，中括号内的键一定是整数。比如，$arr[0]='苹果';

第二种：用array()创建一个空数组，使用=>符号来分隔键和值，左侧表示键，右侧表示值。当然，索引数组中，键一定是整数。比如，array('0'=>'苹果');

第三种：用array()创建一个空数组，直接在数组里用英文的单引号'或者英文的双引号"赋值，数组会默认建立从0开始的整数的键。比如array('苹果');这个数组相当于array('0'=>'苹果');

## for循环访问索引数组里的值

for循环可以将数组里的所有值都访问到，下面我们展示下，用for循环访问索引数组里的值。

例如：

$fruit=array('苹果','香蕉','菠萝');

for($i=0; $i<3; $i++){

echo '<br>数组第'.$i.'值是：'.$fruit[$i];

}

## foreach循环访问索引数组里的值

foreach循环可以将数组里的所有值都访问到，下面我们展示下，用foreach循环访问索引数组里的值。

例如：

$fruit=array('苹果','香蕉','菠萝');

foreach($fruit as $k=>$v){

echo '<br>第'.$k.'值是：'.$v;

}

## 访问关联数组内容

用数组变量的名字后跟中括号+键的方式来访问数组中的值，键使用单引号或者双引号括起来。

比如：

$fruit = array('apple'=>"苹果",'banana'=>"香蕉",'pineapple'=>"菠萝");

$fruit0 = **$fruit['banana']**;

print\_r($fruit0);

上面代码的作用：先把一个数组$fruit中键为banana字符串的值取出来然后赋给一个另一个变量$fruit0，最后输出变量值。

## foreach循环访问关联数组里的值

foreach循环可以将数组里的所有值都访问到，下面我们展示下，用foreach循环访问关联数组里的值。

例如：

$fruit=array('apple'=>"苹果",'banana'=>"香蕉",'pineapple'=>"菠萝");

foreach($fruit as $k=>$v){

echo '<br>水果的英文键名：'.$k.'，对应的值是：'.$v;

}

## 返回值

使用return关键字可以使函数返回值，可以返回包括数组和对象的任意类型，如果省略了 return，则默认返回值为 NULL。

function add($a) {

return $a+1;

}

$b = add(1);

返回语句会立即中止函数的运行，并且将控制权交回调用该函数的代码行，因此下面函数的返回值跟上面的函数是一样的。

function add($a) {

return $a+1;

$a = 10;

return $a+20;

}

$b = add(1);

函数不能返回多个值，但可以通过返回一个数组来得到类似的效果。

function numbers() {

return array(1, 2, 3);

}

list ($one, $two, $three) = numbers();

## 可变函数

所谓可变函数，即通过变量的值来调用函数，因为变量的值是可变的，所以可以通过改变一个变量的值来实现调用不同的函数。经常会用在回调函数、函数列表，或者根据动态参数来调用不同的函数。可变函数的调用方法为变量名加括号。

function name() {

echo 'jobs';

}

$func = 'name';

$func(); //调用可变函数

可变函数也可以用在对象的方法调用上。

class book {

function getName() {

return 'bookname';

}

}

$func = 'getName';

$book = new book();

$book->$func();

## 判断函数是否存在

当我们创建了自定义函数，并且了解了可变函数的用法，为了确保程序调用的函数是存在的，经常会先使用function\_exists判断一下函数是否存在。同样的method\_exists可以用来检测类的方法是否存在。

function func() {

}

if (function\_exists('func')){

echo 'exists';

}

类是否定义可以使用class\_exists。

class MyClass{

}

// 使用前检查类是否存在

if (class\_exists('MyClass')) {

$myclass = new MyClass();

}

PHP中有很多这类的检查方法，例如文件是否存在file\_exists等。

$filename = 'test.txt';

if (!file\_exists($filename)) {

echo $filename . ' not exists.';

}

## 类和对象

类是面向对象程序设计的基本概念，通俗的理解类就是对现实中某一个种类的东西的抽象， 比如汽车可以抽象为一个类，汽车拥有名字、轮胎、速度、重量等属性，可以有换挡、前进、后退等操作方法。

通常定义一个汽车类的方法为：

class Car {

$name = '汽车';

function getName() {

return $this->name;

}

}

类是一类东西的结构描述，而对象则是一类东西的一个具体实例，例如汽车这个名词可以理解为汽车的总类，但这辆汽车则是一个具体的汽车对象。

对象通过new关键字进行实例化：

$car = new Car();

echo $car->getName();

类与对象看起来比较相似，但实际上有本质的区别，类是抽象的概念，对象是具体的实例。类可以使程序具有可重用性。

## 类的属性

在类中定义的变量称之为属性，通常属性跟数据库中的字段有一定的关联，因此也可以称作“字段”。属性声明是由关键字 public，protected 或者 private 开头，后面跟一个普通的变量声明来组成。属性的变量可以设置初始化的默认值，默认值必须是常量。

访问控制的关键字代表的意义为：

public：公开的  
protected：受保护的  
private：私有的

class Car {

//定义公共属性

public $name = '汽车';

//定义受保护的属性

protected $corlor = '白色';

//定义私有属性

private $price = '100000';

}

默认都为public，外部可以访问。一般通过->对象操作符来访问对象的属性或者方法，对于静态属性则使用::双冒号进行访问。当在类成员方法内部调用的时候，可以使用$this伪变量调用当前对象的属性。

$car = new Car();

echo $car->name; //调用对象的属性

echo $car->color; //错误 受保护的属性不允许外部调用

echo $car->price; //错误 私有属性不允许外部调用

受保护的属性与私有属性不允许外部调用，在类的成员方法内部是可以调用的。

class Car{

private $price = '1000';

public function getPrice() {

return $this->price; //内部访问私有属性

​ }

}

## 定义类的方法

方法就是在类中的function，很多时候我们分不清方法与函数有什么差别，在面向过程的程序设计中function叫做函数，在面向对象中function则被称之为方法。

同属性一样，类的方法也具有public，protected 以及 private 的访问控制。

访问控制的关键字代表的意义为：  
public：公开的  
protected：受保护的  
private：私有的

我们可以这样定义方法：

class Car {

public function getName() {

return '汽车';

}

​}

$car = new Car();

echo $car->getName();

使用关键字static修饰的，称之为静态方法，静态方法不需要实例化对象，可以通过类名直接调用，操作符为双冒号::。

class Car {

public static function getName() {

return '汽车';

}

​}

echo Car::getName(); //结果为“汽车”

## 构造函数和析构函数

PHP5可以在类中使用**\_\_construct()**定义一个构造函数，具有构造函数的类，会在每次对象创建的时候调用该函数，因此常用来在对象创建的时候进行一些初始化工作。

class Car {

function **\_\_construct()** {

print "构造函数被调用\n";

}

}

$car = new Car(); //实例化的时候 会自动调用构造函数\_\_construct，这里会输出一个字符串

在子类中如果定义了\_\_construct则不会调用父类的\_\_construct，如果需要同时调用父类的构造函数，需要使用parent::\_\_construct()显式的调用。

class Car {

function \_\_construct() {

print "父类构造函数被调用\n";

}

}

class Truck extends Car {

   function \_\_construct() {

       print "子类构造函数被调用\n";

  parent::\_\_construct();

   }

}

$car = new Truck();

同样，PHP5支持析构函数，使用**\_\_destruct()**进行定义，析构函数指的是当某个对象的所有引用被删除，或者对象被显式的销毁时会执行的函数。

class Car {

function \_\_construct() {

print "构造函数被调用 \n";

}

function **\_\_destruct()** {

print "析构函数被调用 \n";

}

}

$car = new Car(); //实例化时会调用构造函数

echo '使用后，准备销毁car对象 \n';

unset($car); //销毁时会调用析构函数

当PHP代码执行完毕以后，会自动回收与销毁对象，因此一般情况下不需要显式的去销毁对象。

## Static静态关键字

静态属性与方法可以在不实例化类的情况下调用，直接使用类名::方法名的方式进行调用。静态属性**不允许**对象使用->操作符调用。目前仅有静态采用此调用方法

class Car {

private static $speed = 10;

public static function getSpeed() {

return self::$speed;

}

}

echo Car::getSpeed(); //调用静态方法

静态方法也可以通过变量来进行动态调用

$func = 'getSpeed';

$className = 'Car';

echo $className::$func(); //动态调用静态方法

静态方法中，$this伪变量不允许使用。可以使用self，parent，static在内部调用静态方法与属性。

class Car {

private static $speed = 10;

public static function getSpeed() {

return self::$speed;

}

public static function speedUp() {

return self::$speed+=10;

}

}

class BigCar extends Car {

public static function start() {

parent::speedUp();

}

}

**BigCar::start();**

echo BigCar::getSpeed();

## 访问控制

前面的小节，我们已经接触过访问控制了，访问控制通过关键字public，protected和private来实现。被定义为公有的类成员可以在任何地方被访问。被定义为受保护的类成员则可以被其自身以及其子类和父类访问。被定义为私有的类成员则只能被其定义所在的类访问。

类属性必须定义为**公有**、**受保护**、**私有**之一。为兼容PHP5以前的版本，如果采用 var 定义，则被视为公有。

class Car {

$speed = 10; //错误 属性必须定义访问控制

public $name; //定义共有属性

}

类中的方法可以被定义为**公有**、**私有**或**受保护**。如果没有设置这些关键字，则该方法默认为**公有**。

class Car {

​ //默认为共有方法

function turnLeft() {

}

}

如果构造函数定义成了私有方法，则不允许直接实例化对象了，这时候一般通过静态方法进行实例化，在设计模式中会经常使用这样的方法来控制对象的创建，比如单例模式只允许有一个全局唯一的对象。

class Car {

private function \_\_construct() {

echo 'object create';

}

private static $\_object = null;

public static function getInstance() {

if (empty(self::$\_object)) {

self::$\_object = new Car(); //内部方法可以调用私有方法，因此这里可以创建对象

}

return self::$\_object;

}

}

//$car = new Car(); //这里不允许直接实例化对象

$car = Car::getInstance(); //通过静态方法来获得一个实例

## 对象继承

继承是面向对象程序设计中常用的一个特性，汽车是一个比较大的类，我们也可以称之为基类，除此之外，汽车还分为卡车、轿车、东风、宝马等，因为这些子类具有很多相同的属性和方法，可以采用继承汽车类来共享这些属性与方法，实现代码的复用。

## 重载

PHP中的重载指的是动态的创建属性与方法，是通过魔术方法来实现的。属性的重载通过\_\_set，\_\_get，\_\_isset，\_\_unset来分别实现对不存在属性的赋值、读取、判断属性是否设置、销毁属性。

class Car {

private $ary = array();

public function \_\_set($key, $val) {

$this->ary[$key] = $val;

}

public function \_\_get($key) {

if (isset($this->ary[$key])) {

return $this->ary[$key];

}

return null;

}

public function \_\_isset($key) {

if (isset($this->ary[$key])) {

return true;

}

return false;

}

public function \_\_unset($key) {

unset($this->ary[$key]);

}

}

$car = new Car();

$car->name = '汽车'; //name属性动态创建并赋值

echo $car->name;

方法的重载通过\_\_call来实现，当调用不存在的方法的时候，将会转为参数调用\_\_call方法，当调用不存在的静态方法时会使用\_\_callStatic重载。

class Car {

public $speed = 0;

public function \_\_call($name, $args) {

if ($name == 'speedUp') {

$this->speed += 10;

}

}

}

$car = new Car();

$car->speedUp(); //调用不存在的方法会使用重载

echo $car->speed;

## 对象的高级特性

对象比较，当同一个类的两个实例的所有属性都相等时，可以使用比较运算符==进行判断，当需要判断两个变量是否为同一个对象的引用时，可以使用全等运算符===进行判断。

class Car {

}

$a = new Car();

$b = new Car();

if ($a == $b) echo '=='; //true

if ($a === $b) echo '==='; //false

对象复制，在一些特殊情况下，可以通过关键字clone来复制一个对象，这时\_\_clone方法会被调用，通过这个魔术方法来设置属性的值。

class Car {

public $name = 'car';

public function \_\_clone() {

$obj = new Car();

$obj->name = $this->name;

}

}

$a = new Car();

$a->name = 'new car';

$b = clone $a;

var\_dump($b);

对象序列化，可以通过serialize方法将对象序列化为字符串，用于存储或者传递数据，然后在需要的时候通过unserialize将字符串反序列化成对象进行使用。

class Car {

public $name = 'car';

}

$a = new Car();

$str = serialize($a); //对象序列化成字符串

echo $str.'<br>';

$b = unserialize($str); //反序列化为对象

var\_dump($b);

## 字符串介绍

PHP开发中，我们遇到最多的可能就是字符串。

字符串变量用于包含字符串的值。

一个字符串 通过下面的3种方法来定义：

1、单引号  
2、双引号  
3、heredoc语法结构

基本用法如下，后面的节还会详细进行介绍使用技巧，这里只介绍用法：

单引号定义的字符串：$hello = 'hello world';

双引号定义的字符串：$hello = "hello world";

heredoc语法结构定义的字符串：

$hello = <<<TAG

hello world

TAG;

## 单引号和双引号的区别

在PHP中，字符串的定义可以使用英文单引号' '，也可以使用英文双引号" "。

但是必须使用同一种单或双引号来定义字符串，如：'Hello World"和"Hello World'为非法的字符串定义。

单引号和双引号到底有啥区别呢？

PHP允许我们在双引号串中直接包含字串变量。

而单引号串中的内容总被认为是普通字符。

比如：

$str='hello';

echo "str is $str"; //运行结果: str is hello

echo 'str is $str'; //运行结果: str is $str

## 去除字符串首尾的空格

PHP中有三个函数可以去掉字符串的空格

trim去除一个字符串两端空格。  
rtrim是去除一个字符串右部空格，其中的r是right的缩写。  
ltrim是去除一个字符串左部空格，其中的l是left的缩写。

例子如下：

echo trim(" 空格 ")."<br>";

echo rtrim(" 空格 ")."<br>";

echo ltrim(" 空格 ")."<br>";

## 获取字符串的长度

php中有一个神奇的函数，可以直接获取字符串的长度，这个函数就是strlen()。

例子如下：

$str = 'hello';

$len = strlen($str);

echo $len;//输出结果是5

strlen函数对于计算**英文**字符是非常的擅长，但是如果有**中文汉字**，要计算长度该怎么办？

可以使用mb\_strlen()函数获取字符串中中文长度。

例子如下：

$str = "我爱你";

echo mb\_strlen($str,"UTF8");//结果：3，此处的UTF8表示中文编码是UTF8格式，中文一般采用UTF8编码

## 字符串的截取

php中有非常多的字符串处理函数，其中就有字符串截取函数。

1、英文字符串的截取函数substr()

函数说明：substr(字符串变量,开始截取的位置，截取个数）

例如：

$str='i love you';

//截取love这几个字母

echo substr($str, 2, 4);//为什么开始位置是2呢，因为substr函数计算字符串位置是从0开始的，也就是0的位置是i,1的位置是空格，l的位置是2。从位置2开始取4个字符，就是love。

2、中文字符串的截取函数mb\_substr()

函数说明：mb\_substr(字符串变量,开始截取的位置，截取个数, 网页编码）

例如：

$str='我爱你，中国';

//截取中国两个字

echo mb\_substr($str, 4, 2, 'utf8');//为什么开始位置是4呢，和上一个例子一样，因为mb\_substr函数计算汉字位置是从0开始的，也就是0的位置是我,1的位置是爱，4的位置是中。从位置4开始取2个汉字，就是中国。中文编码一般是utf8格式

## 查找字符串

如果有一个字符串$str = 'I want to study at imooc';，怎么样找到其中的imooc在哪个位置呢？

查找字符串，我们需要用到PHP的查找字符串函数strpos();

函数说明：strpos(要处理的字符串, 要定位的字符串, 定位的起始位置[可选])

例子：

$str = 'I want to study at imooc';

$pos = strpos($str, 'imooc');

echo $pos;//结果显示19，表示从位置0开始，imooc在第19个位置开始出现

## 替换字符串

如果有一个字符串$str = 'I want to learn js';，怎么样将js字符替换成你想学的php字符呢？

替换字符串，我们需要用到PHP的替换函数**str\_replace**()

函数说明：**str\_replace**(要查找的字符串, 要替换的字符串, 被搜索的字符串, 替换进行计数[可选])

例子：

$str = 'I want to learn js';

$replace = str\_replace('js', 'php', $str);

echo $replace;//结果显示I want to learn php

## 格式化字符串

如果有一个字符串$str = '99.9';，怎么样使这个字符串变成99.90呢？

我们需要用到PHP的格式化字符串函数sprintf()

函数说明：sprintf(格式, 要转化的字符串)

返回：格式化好的字符串

例子：

$str = '99.9';

$result = sprintf('%01.2f', $str);

echo $result;//结果显示99.90

解释下，上面例子中的格式

这个 %01.2f 是什么意思呢？

1、这个 % 符号是开始的意思，写在最前面表示指定格式开始了。 也就是 "起始字符", 直到出现 "转换字符" 为止，就算格式终止。  
2、跟在 % 符号后面的是 0， 是 "填空字元" ，表示如果位置空着就用0来填满。  
3、在 0 后面的是1，这个 1 是规定整个所有的字符串占位要有1位以上(小数点也算一个占位)。  
    如果把 1 改成 6，则 $result的值将为 099.90  
    因为，在小数点后面必须是两位，99.90一共5个占位，现在需要6个占位，所以用0来填满。  
4、在 %01 后面的 .2 （点2） 就很好理解了，它的意思是，小数点后的数字必须占2位。 如果这时候，$str 的值为9.234,则 $result的值将为9.23.  
    为什么4 不见了呢？ 因为在小数点后面，按照上面的规定，必须且仅能占2位。 可是 $str 的值中，小数点后面占了3位，所以，尾数4被去掉了，只剩下 23。  
5、最后，以 f "转换字符" 结尾。

## 字符串的合并与分割

**1、php字符串合并函数implode()**

函数说明：implode(分隔符[可选], 数组)

返回值：把数组元素组合为一个字符串

例子：

$arr = array('Hello', 'World!');

$result = implode('', $arr);

print\_r($result);//结果显示Hello World!

**2、php字符串分隔函数explode()**

函数说明：explode(分隔符[可选], 字符串)

返回值：函数返回由字符串组成的数组

例子：

$str = 'apple,banana';

$result = explode(',', $str);

print\_r($result);//结果显示array('apple','banana')

## 字符串的转义

**php字符串转义函数addslashes()**

函数说明：用于对特殊字符加上转义字符，返回一个字符串

返回值：一个经过转义后的字符串

例子：

$str = "what's your name?";

echo addslashes($str);//输出：what\'s your name?

## 什么叫正则表达式

正则表达式是对字符串进行操作的一种逻辑公式，就是用一些特定的字符组合成一个规则字符串，称之为正则匹配模式。

$p = '/apple/';

$str = "apple banna";

if (preg\_match($p, $str)) {

echo 'matched';

}

其中字符串'/apple/'就是一个正则表达式，他用来匹配源字符串中是否存在apple字符串。

PHP中使用PCRE库函数进行正则匹配，比如上例中的preg\_match用于执行一个正则匹配，常用来判断一类字符模式是否存在。

## 正则表达式的基本语法

PCRE库函数中，正则匹配模式使用分隔符与元字符组成，分隔符可以是非数字、非反斜线、非空格的任意字符。经常使用的分隔符是正斜线(/)、hash符号(#) 以及取反符号(~)，例如：

/foo bar/

#^[^0-9]$#

~php~

如果模式中包含分隔符，则分隔符需要使用反斜杠（\）进行转义。

/http:\/\//

如果模式中包含较多的分割字符，建议更换其他的字符作为分隔符，也可以采用preg\_quote进行转义。

$p = 'http://';

$p = '/'.preg\_quote($p, '/').'/';

echo $p;

分隔符后面可以使用模式修饰符，模式修饰符包括：i, m, s, x等，例如使用i修饰符可以忽略大小写匹配：

$str = "Http://www.imooc.com/";

if (preg\_match('/http/i', $str)) {

echo '匹配成功';

}

## 元字符与转义

正则表达式中具有特殊含义的字符称之为元字符，常用的元字符有：

\ 一般用于转义字符  
^ 断言目标的开始位置(或在多行模式下是行首)  
$ 断言目标的结束位置(或在多行模式下是行尾)  
. 匹配除换行符外的任何字符(默认)  
[ 开始字符类定义  
] 结束字符类定义  
| 开始一个可选分支  
( 子组的开始标记  
) 子组的结束标记  
? 作为量词，表示 0 次或 1 次匹配。位于量词后面用于改变量词的贪婪特性。 (查阅量词)  
\* 量词，0 次或多次匹配  
+ 量词，1 次或多次匹配  
{ 自定义量词开始标记  
} 自定义量词结束标记

//下面的\s匹配任意的空白符，包括空格，制表符，换行符。[^\s]代表非空白符。[^\s]+表示一次或多次匹配非空白符。

$p = '/^我[^\s]+(苹果|香蕉)$/';

$str = "我喜欢吃苹果";

if (preg\_match($p, $str)) {

echo '匹配成功';

}

元字符具有两种使用场景，一种是可以在任何地方都能使用，另一种是只能在方括号内使用，在方括号内使用的有：

\ 转义字符  
^ 仅在作为第一个字符(方括号内)时，表明字符类取反  
- 标记字符范围

其中^在方括号外面，表示断言目标的开始位置，但在方括号内部则代表字符类取反，方括号内的减号-可以标记字符范围，例如0-9表示0到9之间的所有数字。

//下面的\w匹配字母或数字或下划线。

$p = '/[\w\.\-]+@[a-z0-9\-]+\.(com|cn)/';

$str = "我的邮箱是Spark.eric@imooc.com";

preg\_match($p, $str, $match);

echo $match[0];

## 贪婪模式与懒惰模式

正则表达式中每个元字符匹配一个字符，当使用+之后将会变的贪婪，它将匹配尽可能多的字符，但使用问号?字符时，它将尽可能少的匹配字符，既是懒惰模式。

贪婪模式：在可匹配与可不匹配的时候，优先匹配

//下面的\d表示匹配数字

$p = '/\d+\-\d+/';

$str = "我的电话是010-12345678";

preg\_match($p, $str, $match);

echo $match[0]; //结果为：010-12345678

懒惰模式：在可匹配与可不匹配的时候，优先不匹配

$p = '/\d?\-\d?/';

$str = "我的电话是010-12345678";

preg\_match($p, $str, $match);

echo $match[0]; //结果为：0-1

当我们确切的知道所匹配的字符长度的时候，可以使用{}指定匹配字符数

$p = '/\d{3}\-\d{8}/';

$str = "我的电话是010-12345678";

preg\_match($p, $str, $match);

echo $match[0]; //结果为：010-12345678

## 使用正则表达式进行匹配

使用正则表达式的目的是为了实现比字符串处理函数更加灵活的处理方式，因此跟字符串处理函数一样，其主要用来判断子字符串是否存在、字符串替换、分割字符串、获取模式子串等。

PHP使用PCRE库函数来进行正则处理，通过设定好模式，然后调用相关的处理函数来取得匹配结果。

preg\_match用来执行一个匹配，可以简单的用来判断模式是否匹配成功，或者取得一个匹配结果，他的返回值是匹配成功的次数0或者1，在匹配到1次以后就会停止搜索。

$subject = "abcdef";

$pattern = '/def/';

preg\_match($pattern, $subject, $matches);

print\_r($matches); //结果为：Array ( [0] => def )

上面的代码简单的执行了一个匹配，简单的判断def是否能匹配成功，但是正则表达式的强大的地方是进行模式匹配，因此更多的时候，会使用模式：

$subject = "abcdef";

$pattern = '/a(.\*?)d/';

preg\_match($pattern, $subject, $matches);

print\_r($matches); //结果为：Array ( [0] => abcd [1] => bc )

通过正则表达式可以匹配一个模式，得到更多的有用的数据。

## 查找所有匹配结果

preg\_match只能匹配一次结果，但很多时候我们需要匹配所有的结果，preg\_match\_all可以循环获取一个列表的匹配结果数组。

$p = "|<[^>]+>(.\*?)</[^>]+>|i";

$str = "<b>example: </b><div align=left>this is a test</div>";

preg\_match\_all($p, $str, $matches);

print\_r($matches);

可以使用preg\_match\_all匹配一个表格中的数据：

$p = "/<tr><td>(.\*?)<\/td>\s\*<td>(.\*?)<\/td>\s\*<\/tr>/i";

$str = "<table> <tr><td>Eric</td><td>25</td></tr> <tr><td>John</td><td>26</td></tr> </table>";

preg\_match\_all($p, $str, $matches);

print\_r($matches);

$matches结果排序为$matches[0]保存完整模式的所有匹配, $matches[1] 保存第一个子组的所有匹配，以此类推。

## 正则表达式的搜索和替换

正则表达式的搜索与替换在某些方面具有重要用途，比如调整目标字符串的格式，改变目标字符串中匹配字符串的顺序等。

例如我们可以简单的调整字符串的日期格式：

$string = 'April 15, 2014';

$pattern = '/(\w+) (\d+), (\d+)/i';

$replacement = '$3, ${1} $2';

echo preg\_replace($pattern, $replacement, $string); //结果为：2014, April 15

其中${1}与$1的写法是等效的，表示第一个匹配的字串，$2代表第二个匹配的。

通过复杂的模式，我们可以更加精确的替换目标字符串的内容。

$patterns = array ('/(19|20)(\d{2})-(\d{1,2})-(\d{1,2})/',

'/^\s\*{(\w+)}\s\*=/');

$replace = array ('\3/\4/\1\2', '$\1 =');//\3等效于$3,\4等效于$4，依次类推

echo preg\_replace($patterns, $replace, '{startDate} = 1999-5-27'); //结果为：$startDate = 5/27/1999

//详细解释下结果：(19|20)表示取19或者20中任意一个数字，(\d{2})表示两个数字，(\d{1,2})表示1个或2个数字，(\d{1,2})表示1个或2个数字。^\s\*{(\w+)\s\*=}表示以任意空格开头的，并且包含在{}中的字符，并且以任意空格结尾的，最后有个=号的。

用正则替换来去掉多余的空格与字符：

$str = 'one two';

$str = preg\_replace('/\s+/', ' ', $str);

echo $str; // 结果改变为'one two'

## 正则匹配常用案例

正则匹配常用在表单验证上，一些字段会有一定的格式要求，比如用户名一般都要求必须是字母、数字或下划线组成，邮箱、电话等也都有自己的规则，因此使用正则表达式可以很好的对这些字段进行验证。

我们通过案例来看一下一般的用户注册页，都怎样对字段进行验证。

## 设置cookie

PHP设置Cookie最常用的方法就是使用setcookie函数，setcookie具有7个可选参数，我们常用到的为前5个：

name（ Cookie名）可以通过$\_COOKIE['name'] 进行访问  
value（Cookie的值）  
expire（过期时间）Unix时间戳格式，默认为0，表示浏览器关闭即失效  
path（有效路径）如果路径设置为'/'，则整个网站都有效  
domain（有效域）默认整个域名都有效，如果设置了'www.imooc.com',则只在www子域中有效

$value = 'test';

setcookie("TestCookie", $value);

setcookie("TestCookie", $value, time()+3600); //有效期一小时

setcookie("TestCookie", $value, time()+3600, "/path/", "imooc.com"); //设置路径与域

PHP中还有一个设置Cookie的函数setrawcookie，setrawcookie跟setcookie基本一样，唯一的不同就是value值不会自动的进行urlencode，因此在需要的时候要手动的进行urlencode。

setrawcookie('cookie\_name', rawurlencode($value), time()+60\*60\*24\*365);

因为Cookie是通过HTTP标头进行设置的，所以也可以直接使用header方法进行设置。

header("Set-Cookie:cookie\_name=value");

## cookie的删除与过期时间

通过前面的章节，我们了解了设置cookie的函数，但是我们却发现php中没有删除Cookie的函数，在PHP中删除cookie也是采用setcookie函数来实现。

setcookie('test', '', time()-1);

可以看到将cookie的过期时间设置到当前时间之前，则该cookie会自动失效，也就达到了删除cookie的目的。之所以这么设计是因为cookie是通过HTTP的标头来传递的，客户端根据服务端返回的Set-Cookie段来进行cookie的设置，如果删除cookie需要使用新的Del-Cookie来实现，则HTTP头就会变得复杂，实际上仅通过Set-Cookie就可以简单明了的实现Cookie的设置、更新与删除。

了解原理以后，我们也可以直接通过header来删除cookie。

header("Set-Cookie:test=1393832059; expires=".gmdate('D, d M Y H:i:s \G\M\T', time()-1));

这里用到了gmdate，用来生成格林威治标准时间，以便排除时差的影响。

## cookie的有效路径

cookie中的路径用来控制设置的cookie在哪个路径下有效，默认为'/'，在所有路径下都有，当设定了其他路径之后，则只在设定的路径以及子路径下有效，例如：

setcookie('test', time(), 0, '/path');

上面的设置会使test在/path以及子路径/path/abc下都有效，但是在根目录下就读取不到test的cookie值。

一般情况下，大多是使用所有路径的，只有在极少数有特殊需求的时候，会设置路径，这种情况下只在指定的路径中才会传递cookie值，可以节省数据的传输，增强安全性以及提高性能。

当我们设置了有效路径的时候，不在当前路径的时候则看不到当前cookie。

setcookie('test', '1',0, '/path');

var\_dump($\_COOKIE['test']);

## session与cookie的异同

cookie将数据存储在客户端，建立起用户与服务器之间的联系，通常可以解决很多问题，但是cookie仍然具有一些局限：

cookie相对不是太安全，容易被盗用导致cookie欺骗  
单个cookie的值最大只能存储4k  
每次请求都要进行网络传输，占用带宽

session是将用户的会话数据存储在服务端，没有大小限制，通过一个session\_id进行用户识别，PHP默认情况下session id是通过cookie来保存的，因此从某种程度上来说，seesion依赖于cookie。但这不是绝对的，session id也可以通过参数来实现，只要能将session id传递到服务端进行识别的机制都可以使用session。

<?php

//开始使用session

session\_start();

//设置一个session

$\_SESSION['test'] = time();

//显示当前的session\_id

echo "session\_id:".session\_id();

echo "<br>";

//读取session值

echo $\_SESSION['test'];

//销毁一个session

unset($\_SESSION['test']);

echo "<br/>";

var\_dump($\_SESSION);

## 使用session

在PHP中使用session非常简单，先执行session\_start方法开启session，然后通过全局变量$\_SESSION进行session的读写。

session\_start();

$\_SESSION['test'] = time();

var\_dump($\_SESSION);

session会自动的对要设置的值进行encode与decode，因此session可以支持任意数据类型，包括数据与对象等。

session\_start();

$\_SESSION['ary'] = array('name' => 'jobs');

$\_SESSION['obj'] = new stdClass();

var\_dump($\_SESSION);

默认情况下，session是以文件形式存储在服务器上的，因此当一个页面开启了session之后，会独占这个session文件，这样会导致当前用户的其他并发访问无法执行而等待。可以采用缓存或者数据库的形式存储来解决这个问题，这个我们会在一些高级的课程中讲到。

## 删除与销毁session

删除某个session值可以使用PHP的unset函数，删除后就会从全局变量$\_SESSION中去除，无法访问。

session\_start();

$\_SESSION['name'] = 'jobs';

unset($\_SESSION['name']);

echo $\_SESSION['name']; //提示name不存在

如果要删除所有的session，可以使用session\_destroy函数销毁当前session，session\_destroy会删除所有数据，但是session\_id仍然存在。

session\_start();

$\_SESSION['name'] = 'jobs';

$\_SESSION['time'] = time();

session\_destroy();

值得注意的是，session\_destroy并不会立即的销毁全局变量$\_SESSION中的值，只有当下次再访问的时候，$\_SESSION才为空，因此如果需要立即销毁$\_SESSION，可以使用unset函数。

session\_start();

$\_SESSION['name'] = 'jobs';

$\_SESSION['time'] = time();

unset($\_SESSION);

session\_destroy();

var\_dump($\_SESSION); //此时已为空

如果需要同时销毁cookie中的session\_id，通常在用户退出的时候可能会用到，则还需要显式的调用setcookie方法删除session\_id的cookie值。

## 使用session来存储用户的登录信息

session可以用来存储多种类型的数据，因此具有很多的用途，常用来存储用户的登录信息，购物车数据，或者一些临时使用的暂存数据等。

用户在登录成功以后，通常可以将用户的信息存储在session中，一般的会单独的将一些重要的字段单独存储，然后所有的用户信息独立存储。

$\_SESSION['uid'] = $userinfo['uid'];

$\_SESSION['userinfo'] = $userinfo;

一般来说，登录信息既可以存储在sessioin中，也可以存储在cookie中，他们之间的差别在于session可以方便的存取多种数据类型，而cookie只支持字符串类型，同时对于一些安全性比较高的数据，cookie需要进行格式化与加密存储，而session存储在服务端则安全性较高。

<?php

session\_start();

//假设用户登录成功获得了以下用户数据

$userinfo = array(

'uid' => 10000,

'name' => 'spark',

'email' => 'spark@imooc.com',

'sex' => 'man',

'age' => '18'

);

header("content-type:text/html; charset=utf-8");

/\* 将用户信息保存到session中 \*/

$\_SESSION['uid'] = $userinfo['uid'];

$\_SESSION['name'] = $userinfo['name'];

$\_SESSION['userinfo'] = $userinfo;

//\* 将用户数据保存到cookie中的一个简单方法 \*/

$secureKey = 'imooc'; //加密密钥

$str = serialize($userinfo); //将用户信息序列化

//用户信息加密前

$str = base64\_encode(mcrypt\_encrypt(MCRYPT\_RIJNDAEL\_256, md5($secureKey), $str, MCRYPT\_MODE\_ECB));

//用户信息加密后

//将加密后的用户数据存储到cookie中

setcookie('userinfo', $str);

//当需要使用时进行解密

$str = mcrypt\_decrypt(MCRYPT\_RIJNDAEL\_256, md5($secureKey), base64\_decode($str), MCRYPT\_MODE\_ECB);

$uinfo = unserialize($str);

echo "解密后的用户信息：<br>";

print\_r($uinfo);

## 读取文件内容

PHP具有丰富的文件操作函数，最简单的读取文件的函数为file\_get\_contents，可以将整个文件全部读取到一个字符串中。

$content = file\_get\_contents('./test.txt');

file\_get\_contents也可以通过参数控制读取内容的开始点以及长度。

$content = file\_get\_contents('./test.txt', null, null, 100, 500);

PHP也提供类似于C语言操作文件的方法，使用fopen，fgets，fread等方法，fgets可以从文件指针中读取一行，freads可以读取指定长度的字符串。

$fp = fopen('./text.txt', 'rb');

while(!feof($fp)) {

echo fgets($fp); //读取一行

}

fclose($fp);

$fp = fopen('./text.txt', 'rb');

$contents = '';

while(!feof($fp)) {

$contents .= fread($fp, 4096); //一次读取4096个字符

}

fclose($fp);

使用fopen打开的文件，最好使用fclose关闭文件指针，以避免文件句柄被占用。

## 判断文件是否存在

一般情况下在对文件进行操作的时候需要先判断文件是否存在，PHP中常用来判断文件存在的函数有两个is\_file与file\_exists.

$filename = './test.txt';

if (file\_exists($filename)) {

echo file\_get\_contents($filename);

}

如果只是判断文件存在，使用file\_exists就行，file\_exists不仅可以判断文件是否存在，同时也可以判断目录是否存在，从函数名可以看出，is\_file是确切的判断给定的路径是否是一个文件。

$filename = './test.txt';

if (is\_file($filename)) {

echo file\_get\_contents($filename);

}

更加精确的可以使用is\_readable与is\_writeable在文件是否存在的基础上，判断文件是否可读与可写。

$filename = './test.txt';

if (is\_writeable($filename)) {

file\_put\_contents($filename, 'test');

}

if (is\_readable($filename)) {

echo file\_get\_contents($filename);

}

## 写入内容到文件

与读取文件对应，PHP写文件也具有两种方式，最简单的方式是采用file\_put\_contents。

$filename = './test.txt';

$data = 'test';

file\_put\_contents($filename, $data);

上例中，$data参数可以是一个一维数组，当$data是数组的时候，会自动的将数组连接起来，相当于$data=implode('', $data);

同样的，PHP也支持类似C语言风格的操作方式，采用fwrite进行文件写入。

$fp = fopen('./test.txt', 'w');

fwrite($fp, 'hello');

fwrite($fp, 'world');

fclose($fp);

## 取得文件的修改时间

文件有很多元属性，包括：文件的所有者、创建时间、修改时间、最后的访问时间等。

fileowner：获得文件的所有者

filectime：获取文件的创建时间

filemtime：获取文件的修改时间

fileatime：获取文件的访问时间

其中最常用的是文件的修改时间，通过文件的修改时间，可以判断文件的时效性，经常用在静态文件或者缓存数据的更新。

$mtime = filemtime($filename);

echo '修改时间：'.date('Y-m-d H:i:s', filemtime($filename));

<?php

$filename = '/data/webroot/usercode/code/resource/test.txt';

echo '所有者：'.fileowner($filename).'<br>';

echo '创建时间：'.filectime($filename).'<br>';

echo '修改时间：'.filemtime($filename).'<br>';

echo '最后访问时间：'.fileatime($filename).'<br>';

//给$mtime赋值为文件的修改时间

$mtime = filemtime($filename);

//通过计算时间差 来判断文件内容是否有效

if (time() - $mtime > 3600) {

echo '<br>缓存已过期';

} else {

echo file\_get\_contents($filename);

}

## 取得文件的大小

通过filesize函数可以取得文件的大小，文件大小是以字节数表示的。

$filename = '/data/webroot/usercode/code/resource/test.txt';

$size = filesize($filename);

如果要转换文件大小的单位，可以自己定义函数来实现。

function getsize($size, $format = 'kb') {

$p = 0;

if ($format == 'kb') {

$p = 1;

} elseif ($format == 'mb') {

$p = 2;

} elseif ($format == 'gb') {

$p = 3;

}

$size /= pow(1024, $p);

return number\_format($size, 3);

}

$filename = '/data/webroot/usercode/code/resource/test.txt';

$size = filesize($filename);

$size = getsize($size, 'kb'); //进行单位转换

echo $size.'kb';

值得注意的是，没法通过简单的函数来取得目录的大小，目录的大小是该目录下所有子目录以及文件大小的总和，因此需要通过递归的方法来循环计算目录的大小。

## 删除文件

跟Unix系统命令类似，PHP使用unlink函数进行文件删除。

unlink($filename);

删除文件夹使用rmdir函数，文件夹必须为空，如果不为空或者没有权限则会提示失败。

rmdir($dir);

如果文件夹中存在文件，可以先循环删除目录中的所有文件，然后再删除该目录，循环删除可以使用glob函数遍历所有文件。

foreach (glob("\*") as $filename) {

unlink($filename);

}

## 取得当前的Unix时间戳

UNIX 时间戳（英文叫做：timestamp）是 PHP 中关于时间与日期的一个很重要的概念，它表示从 1970年1月1日 00:00:00 到当前时间的秒数之和。

PHP提供了内置函数 time() 来取得服务器当前时间的时间戳。那么获取当前的UNIX时间戳就很简单了。

$time = time();

echo $time;//1396193923，这个数字表示从1970年1月1日 00:00:00 到我输出这个脚本时经历了1396193923秒

## 取得日期的Unix时间戳

UNIX 时间戳（英文叫做：timestamp）是 PHP 中关于时间与日期的一个很重要的概念，它表示从 1970年1月1日 00:00:00 到当前时间的秒数之和。

PHP提供了内置函数strtotime实现功能：获取某个日期的时间戳，或获取某个时间的时间戳。例如：

echo strtotime('2014-04-29');//1398700800，这个数字表示从1970年1月1日 00:00:00 到2014年4月29号经历了1398700800秒

echo strtotime('2014-04-29 00:00:01');//1398700801，这个数字表示从1970年1月1日 00:00:00 到2014-04-29 00:00:01时经历了1398700801秒

大家发现上面的规律了吗，其实strtotime('2014-04-29')相当于strtotime('2014-04-29 00:00:00')

## 格式化的日期字符串转换为Unix时间戳

strtotime函数预期接受一个包含美国英语日期格式的字符串并尝试将其解析为 Unix 时间戳。

函数说明：strtotime(要解析的时间字符串, 计算返回值的时间戳【默认是当前的时间，可选】)  
返回值：成功则返回时间戳，否则返回 FALSE

比如

echo strtotime("now");//相当于将英文单词now直接等于现在的日期和时间，并把这个日期时间转化为unix时间戳。这个效果跟echo time();一样。

echo strtotime("+1 seconds");//相当于将现在的日期和时间加上了1秒，并把这个日期时间转化为unix时间戳。这个效果跟echo time()+1;一样。

echo strtotime("+1 day");//相当于将现在的日期和时间加上了1天。

echo strtotime("+1 week");//相当于将现在的日期和时间加上了1周。

echo strtotime("+1 week 3 days 7 hours 5 seconds");//相当于将现在的日期和时间加上了1周3天7小时5秒。

## 格式化格林威治（GMT）标准时间

gmdate 函数能格式化一个GMT的日期和时间，返回的是格林威治标准时（GMT）。

举个例子，我们现在所在的中国时区是东八区，领先格林威治时间8个小时，有时候也叫GMT+8，那么服务器运行以下脚本返回的时间应该是这样的：

当前时间假定是2014-05-01 15:15:22

echo date('Y-m-d H:i:s', time()); //输出为：2014-05-01 15:15:22

echo gmdate('Y-m-d H:i:s', time()); //输出为：2014-05-01 07:15:22 因为格林威治时间是现在中国时区的时间减去8个小时，所以相对于现在时间要少8个小时

date\_default\_timezone\_set("Asia/Shanghai");

GD库

## 抛出一个异常

从PHP5开始，PHP支持异常处理，异常处理是面向对象一个重要特性，PHP代码中的异常通过throw抛出，异常抛出之后，后面的代码将不会再被执行。

既然抛出异常会中断程序执行，那么为什么还需要使用异常处理？

异常抛出被用于在遇到未知错误，或者不符合预先设定的条件时，通知客户程序，以便进行其他相关处理，不至于使程序直接报错中断。

当代码中使用了try catch的时候，抛出的异常会在catch中捕获，否则会直接中断。

1、基本语法  
        try{  
            //可能出现错误或异常的代码  
            //catch表示捕获，Exception是php已定义好的异常类  
        } catch(Exception $e){  
            //对异常处理，方法：  
                //1、自己处理  
                //2、不处理，将其再次抛出  
        }  
2、处理程序应当包括：  
Try - 使用异常的函数应该位于 "try"  代码块内。如果没有触发异常，则代码将照常继续执行。但是如果异常被触发，会抛出一个异常。  
Throw - 这里规定如何触发异常。注意：每一个 "throw" 必须对应至少一个 "catch"，当然可以对应多个"catch"  
Catch - "catch" 代码块会捕获异常，并创建一个包含异常信息的对象。

//创建可抛出一个异常的函数

function checkNum($number){

if($number>1){

throw new Exception("异常提示-数字必须小于等于1");

}

return true;

}

//在 "try" 代码块中触发异常

try{

checkNum(2);

//如果异常被抛出，那么下面一行代码将不会被输出

echo '如果能看到这个提示，说明你的数字小于等于1';

}catch(Exception $e){

//捕获异常

echo '捕获异常: ' .$e->getMessage();

}

上面代码将获得类似这样一个错误：

捕获异常:: 异常提示-数字必须小于等于1

例子解释：

上面的代码抛出了一个异常，并捕获了它：

创建 checkNum() 函数。它检测数字是否大于 1。如果是，则抛出一个异常。  
在 "try" 代码块中调用 checkNum() 函数。  
checkNum() 函数中的异常被抛出  
"catch" 代码块接收到该异常，并创建一个包含异常信息的对象 ($e)。  
通过从这个 exception 对象调用 $e->getMessage()，输出来自该异常的错误消息

## 异常处理类

PHP具有很多异常处理类，其中Exception是所有异常处理的基类。

Exception具有几个基本属性与方法，其中包括了：

message 异常消息内容  
code 异常代码  
file 抛出异常的文件名  
line 抛出异常在该文件的行数

其中常用的方法有：

getTrace 获取异常追踪信息  
getTraceAsString 获取异常追踪信息的字符串  
getMessage 获取出错信息

如果必要的话，可以通过继承Exception类来建立自定义的异常处理类。

//自定义的异常类，继承了PHP的异常基类Exception

class MyException extends Exception {

function getInfo() {

return '自定义错误信息';

}

}

try {

//使用异常的函数应该位于 "try" 代码块内。如果没有触发异常，则代码将照常继续执行。但是如果异常被触发，会抛出一个异常。

throw new MyException('error');//这里规定如何触发异常。注意：每一个 "throw" 必须对应至少一个 "catch"，当然可以对应多个"catch"

} catch(Exception $e) {//"catch" 代码块会捕获异常，并创建一个包含异常信息的对象

echo $e->getInfo();//获取自定义的异常信息

  echo $e->getMessage();//获取继承自基类的getMessage信息

}

## 捕获异常信息

在了解了异常处理的基本原理之后，我们可以通过try catch来捕获异常，我们将执行的代码放在try代码块中，一旦其中的代码抛出异常，就能在catch中捕获。

这里我们只是通过案例来了解try catch的机制以及异常捕获的方法，在实际应用中，不会轻易的抛出异常，只有在极端情况或者非常重要的情况下，才会抛出异常，抛出异常，可以保障程序的正确性与安全，避免导致不可预知的bug。

一般的异常处理流程代码为：

try {

throw new Exception('wrong');

} catch(Exception $ex) {

    echo 'Error:'.$ex->getMessage().'<br>';

    echo $ex->getTraceAsString().'<br>';

}

echo '异常处理后，继续执行其他代码';

## 获取错误发生的所在行

在异常被捕获之后，我们可以通过异常处理对象获取其中的异常信息，前面我们已经了解捕获方式，以及获取基本的错误信息。

在实际应用中，我们通常会获取足够多的异常信息，然后写入到错误日志中。

通过我们需要将报错的文件名、行号、错误信息、异常追踪信息等记录到日志中，以便调试与修复问题。

<?php

try {

throw new Exception('wrong');

} catch(Exception $ex) {

$msg = 'Error:'.$ex->getMessage()."\n";

$msg.= $ex->getTraceAsString()."\n";

$msg.= '异常行号：'.$ex->getLine()."\n";

$msg.= '所在文件：'.$ex->getFile()."\n";

//将异常信息记录到日志中

file\_put\_contents('error.log', $msg);

}

## PHP支持哪些数据库

PHP通过安装相应的扩展来实现数据库操作，现代应用程序的设计离不开数据库的应用，当前主流的数据库有MsSQL，MySQL，Sybase，Db2，Oracle，PostgreSQL，Access等，这些数据库PHP都能够安装扩展来支持，一般情况下常说的LAMP架构指的是：Linux、Apache、Mysql、PHP，因此Mysql数据库在PHP中的应用非常广泛，我们会在本章中简单的了解Mysql的操作方法。

<?php

if (function\_exists('mysql\_connect')) {

echo 'Mysql扩展已经安装';

}

## 数据库扩展

PHP中一个数据库可能有一个或者多个扩展，其中既有官方的，也有第三方提供的。像Mysql常用的扩展有原生的mysql库，也可以使用增强版的mysqli扩展，还可以使用PDO进行连接与操作。

不同的扩展提供基本相近的操作方法，不同的是可能具备一些新特性，以及操作性能可能会有所不同。

mysql扩展进行数据库连接的方法：

$link = mysql\_connect('mysql\_host', 'mysql\_user', 'mysql\_password');

mysqli扩展：

$link = mysqli\_connect('mysql\_host', 'mysql\_user', 'mysql\_password');

PDO扩展

$dsn = 'mysql:dbname=testdb;host=127.0.0.1';

$user = 'dbuser';

$password = 'dbpass';

$dbh = new PDO($dsn, $user, $password);

## 连接MySQL数据库

PHP要对数据库进行操作，首先要做的是与数据库建立连接，通常我们使用mysql\_connect函数进行数据库连接，该函数需要指定数据库的地址，用户名及密码。

$host = 'localhost';

$user = 'code1';

$pass = '';

$link = mysql\_connect($host, $user, $pass);

PHP连接数据库的方式类似于直接在命令行下通过进行连接，类似：mysql -hlocalhost -ucode1 -p，当连接成功以后，我们需要选择一个操作的数据库，通过mysql\_select\_db函数来选择数据库。

mysql\_select\_db('code1');

通常我们会先设置一下当前连接使用的字符编码，一般的我们会使用utf8编码。

mysql\_query("set names 'utf8'");

通过上面的步骤，我们就与数据库建立了连接，可以进行数据操作了。

## 执行MySQL查询

在数据库建立连接以后就可以进行查询，采用mysql\_query加sql语句的形式向数据库发送查询指令。

$res = mysql\_query('select \* from user limit 1');

对于查询类的语句会返回一个资源句柄（resource），可以通过该资源获取查询结果集中的数据。

$row = mysql\_fetch\_array($res);

var\_dump($row);

默认的，PHP使用最近的数据库连接执行查询，但如果存在多个连接的情况，则可以通过参数指令从那个连接中进行查询。

$link1 = mysql\_connect('127.0.0.1', 'code1', '');

$link2 = mysql\_connect('127.0.0.1', 'code1', '', true); //开启一个新的连接

$res = mysql\_query('select \* from user limit 1', $link1); //从第一个连接中查询数据

## 插入新数据到MySQL中

当我们了解了如何使用mysql\_query进行数据查询以后，那么类似的，插入数据其实也是通过执行一个sql语句来实现，例如：

$sql = "insert into user(name, age, class) values('李四', 18, '高三一班')";

mysql\_query($sql); //执行插入语句

通常数据都是存储在变量或者数组中，因此sql语句需要先进行字符串拼接得到。

$name = '李四';

$age = 18;

$class = '高三一班';

$sql = "insert into user(name, age, class) values('$name', '$age', '$class')";

mysql\_query($sql); //执行插入语句

在mysql中，执行插入语句以后，可以得到自增的主键id,通过PHP的mysql\_insert\_id函数可以获取该id。

$uid = mysql\_insert\_id();

这个id的作用非常大，通常可以用来判断是否插入成功，或者作为关联ID进行其他的数据操作。

## 取得数据查询结果

通过前面的章节，我们发现PHP操作数据库跟MySql客户端上操作极为相似，先进行连接，然后执行sql语句，再然后获取我们想要的结果集。

PHP有多个函数可以获取数据集中的一行数据，最常用的是mysql\_fetch\_array，可以通过设定参数来更改行数据的下标，默认的会包含数字索引的下标以及字段名的关联索引下标。

$sql = "select \* from user limit 1";

$result = mysql\_query($sql);

$row = mysql\_fetch\_array($result);

可以通过设定参数MYSQL\_NUM只获取数字索引数组，等同于mysql\_fetch\_row函数，如果设定参数为MYSQL\_ASSOC则只获取关联索引数组，等同于mysql\_fetch\_assoc函数。

$row = mysql\_fetch\_row($result);

$row = mysql\_fetch\_array($result, MYSQL\_NUM); //这两个方法获取的数据是一样的

$row = mysql\_fetch\_assoc($result);

$row = mysql\_fetch\_array($result, MYSQL\_ASSOC);

如果要获取数据集中的所有数据，我们通过循环来遍历整个结果集。

$data = array();

while ($row = mysql\_fetch\_array($result)) {

$data[] = $row;

}

## 查询分页数据

上一节中，我们了解到通过循环可以获取一个查询的所有数据，在实际应用中，我们并不希望一次性获取数据表中的所有数据，那样性能会非常的低，因此会使用翻页功能，每页仅显示10条或者20条数据。

通过mysql的limit可以很容易的实现分页，limit m,n表示从m行后取n行数据，在PHP中我们需要构造m与n来实现获取某一页的所有数据。

假定当前页为$page，每页显示$n条数据，那么m为当前页前面所有的数据，既$m = ($page-1) \* $n，在知道了翻页原理以后，那么我们很容易通过构造SQL语句在PHP中实现数据翻页。

$page = 2;

$n = 2;

$m = ($page - 1) \* $n;

$sql = "select \* from user limit $m, $n";

$result = mysql\_query($sql);

//循环获取当前页的数据

$data = array();

while ($row = mysql\_fetch\_assoc($result)) {

$data[] = $row;

}

在上面的例子中，我们使用了$m与$n变量来表示偏移量与每页数据条数，但我们推荐使用更有意义的变量名来表示，比如$pagesize, $start, $offset等，这样更容易理解，有助于团队协作开发。

## 更新与删除数据

数据的更新与删除相对比较简单，只需要构建好相应的sql语句，然后调用mysql\_query执行就能完成相应的更新与删除操作。

$sql = "update user set name = '曹操' where id=2 limit 1";

if (mysql\_query($sql)) {

echo '更新成功';

}

同样的删除可以使用类似以下的代码：

$sql = "delete from user where id=2 limit 1";

if (mysql\_query($sql)) {

echo '删除成功';

}

对于删除与更新操作，可以通过mysql\_affected\_rows函数来获取更新过的数据行数，如果数据没有变化，则结果为0。

$sql = "update user set name = '曹操' where id=2 limit 1";

if (mysql\_query($sql)) {

echo mysql\_affected\_rows();

}

## 关闭MySQL连接

当数据库操作完成以后，可以使用mysql\_close关闭数据库连接，默认的，当PHP执行完毕以后，会自动的关闭数据库连接。

mysql\_close();

虽然PHP会自动关闭数据库连接，一般情况下已经满足需求，但是在对性能要求比较高的情况下，可以在进行完数据库操作之后尽快关闭数据库连接，以节省资源，提高性能。

在存在多个数据库连接的情况下，可以设定连接资源参数来关闭指定的数据库连接。

$link = mysql\_connect($host, $user, $pass);

mysql\_close($link);

<?php

//连接数据库

mysql\_connect('127.0.0.1', 'code1', '');

mysql\_select\_db('code1');

mysql\_query("set names 'utf8'");

//在这里关闭数据库连接

mysql\_close();