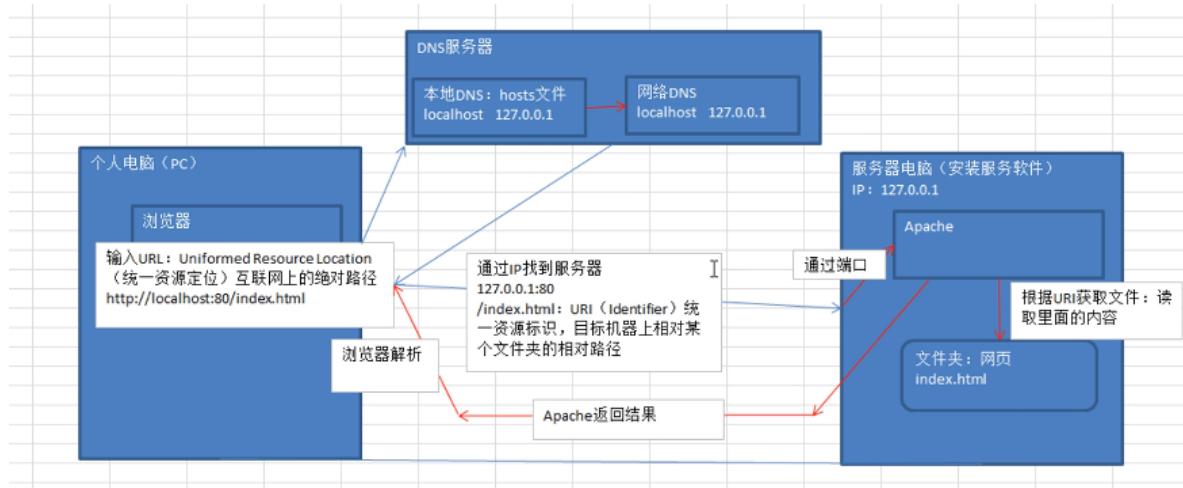


视频：<https://www.bilibili.com/video/BV18x411H7qD?p=3>

# 基础

Web分为两类：静态网站和动态网站。流程：浏览器发起访问->DNS解析域名->服务器电脑->服务软件

## 1、静态网站访问原理：

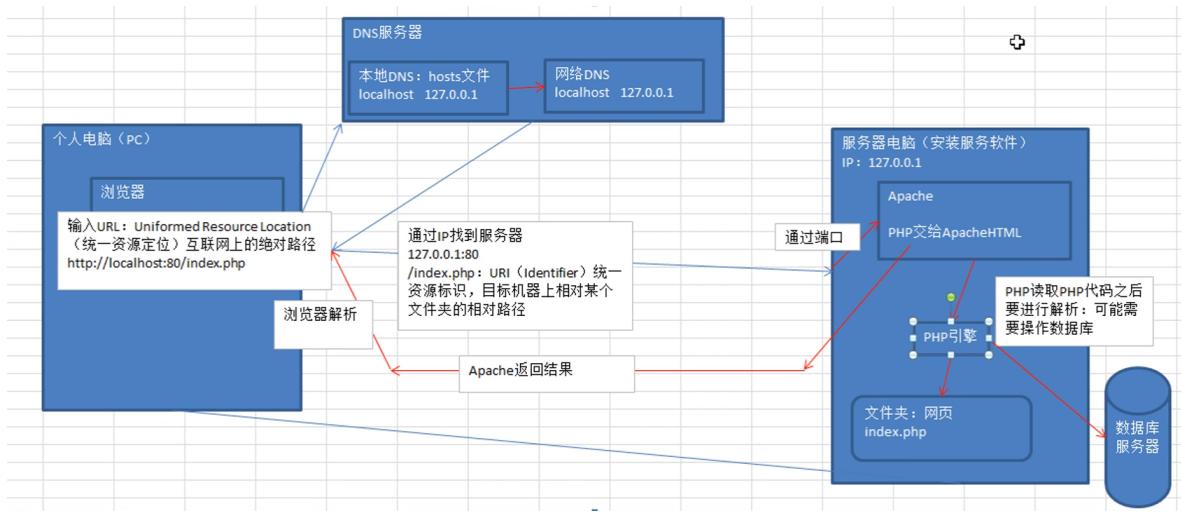


**必备工具**：1、有两台电脑，PC（个人电脑），里面有浏览器；另外一台为服务器（需要有服务软件，假设为APACHE），软件没有数据，故需在里面添加文件夹（准备访问的文件/网页）2、下面蓝线代表两个在同一个互联网下。

**访问流程**：1、在浏览器中需输入URL（uniformed resource location，统一资源定位，即互联网的绝对路径）；2、第一部需要经过URL的解析（DNS，DNS内部包含本地DNS与网络DNS，访问本地DNS无需通过网络，可提高效率。如果本地DNS无法转化IP，就需要网络DNS）。3、DNS将转换后的IP地址返回给浏览器。4、浏览器通过IP地址找到服务器。5、（虽然找到了服务器，但服务器里面资源很多，如何找自己想要的APACHE？）这就需要利用端口，APACHE的端口为80。6、访问APACHE后，即可找到要访问的内容（IP地址后的/index.html也称为URI（uniformed resource identify，统一资源标志。目标机器上相对某个文件夹的相对路径），注即为根目录，相对根目录所以其为相对路径）。7、Apache根据URI读取文件，读取文件的内容（注意只是读取）。8、Apache读取文件后，返回结果。9、浏览器看之前，需要经过浏览器解析，才可展示给用户。（所有人访问的时候，都访问的一样的index文件，若要改变显示的信息，就需要改变index，比较麻烦）

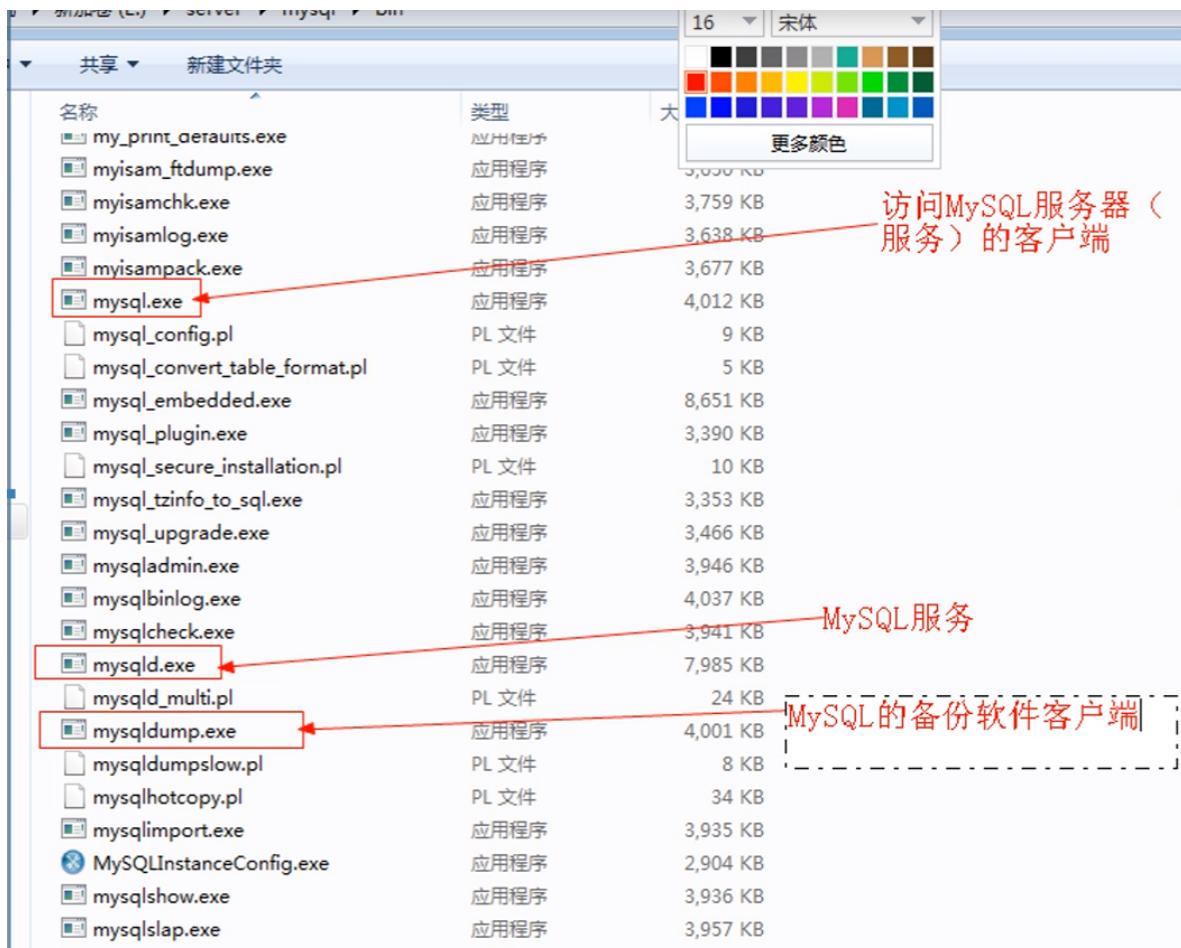
## 2、动态网站访问

区别：相对静态网站访问多了几个步骤：需要服务器端解析；需要数据库



在访问内容时，Apache不能识别PHP（只能识别html），需通过PHP引擎读取PHP代码。读取PHP代码之后需要进行解析，解析过程中可能需要数据库。PHP引擎通过数据库，找到内容以后再返回给Apache。（注意，PHP引擎得到的内容是PHP代码，但交给Apache的是HTML内容，，，因为浏览器只能解析html文件）

### 3、C/S、B/S



**软件设计结构**：C/S、B/S

C/S：Client客户端/Server服务端，用户需要安装客户端产品才能访问服务器，且只能访问一种软件（自己的）

B/S：Browser浏览器/Server服务端，用户只需要安装浏览器，就可以访问所有的服务器（B/S架构服务）。

### 4、MySQL访问流程

Mysql是一款C/S架构的软件，需要通过客户端来访问服务端

**运行：**1、启用MySQL客户端：mysq.exe

2、mysql.exe通过cmd运行（安装配置环境变量后可这样运行）

**操作：**由于qq服务器不在自己的电脑（即客户端与服务器可分离），故MySQL客户端访问服务器需进行寻找匹配：连接认证

a、连接：IP端口和确认。。。如果是本地，均可省略

命令：`-h 主机地址 -localhost ( 或 IP )` localhost是主机名 `-P 端口 -P3306` ( MySQL服务器的端口默认是3306 )

b、认证：通过用户名和密码进入服务器。

`-u 用户名 -uroot ( 不可省略，匿名用户除外 ) -p 密码 -proot`

```
C:\Users\windows>mysql -hlocalhost -P3306 -uroot -proot
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.5.20 MySQL Community Server <GPL>

Copyright (c) 2000, 2011, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> 
```

退出命令行：`\q`

**注意：**通常连接认证的时候不建议密码明文，可在输入-p之后回车，系统会再次让输入密码，此时为密文。

## 5、PHP连接MySQL数据库

PHP本身不具备操作MySQL数据库的能力，需要借助MySQL扩展来实现。

1、PHP加载MySQL扩展：php.ini文件中。（不要用记事本打开）

2、PHP中所有扩展都是在ext的文件夹中，需要指定扩展所在路径：extension\_dir。

3、php.ini已经被Apache重载，故需要重启才可。

## 6、设定系统时区

```
1 ;date.timezone =
2 ; 增加时区
3 date.timezone = PRC
```

## 7、配置虚拟主机

一台服务器若只能部署一个网站，则非常浪费。故需通过其他渠道实现一台主机上部署多个网站。

**虚拟主机**：Virtual machine，并不存在的主机，但可提供真实主机所实现的功能。虚拟主机是将计算机中不同的文件夹进行不同的命名，然后可实现让服务器（Apache）根据用户的需求从不同的文件夹（网站）中读取不同的内容。

**分类**：在Apache中，可将虚拟主机划分为两类

1、基于IP的虚拟主机，一台电脑上有多个IP，每个IP对应一个网站。

原理：电脑默认只有一个IP，因为通常只有一个网卡，但有的电脑可配置多个网卡，每个网卡可绑定一个IP地址。

2、基于域名的虚拟主机：一台主机只有一个IP，但IP下可制作多个网站，但需给每个网站不同的名字。

# 基础语法

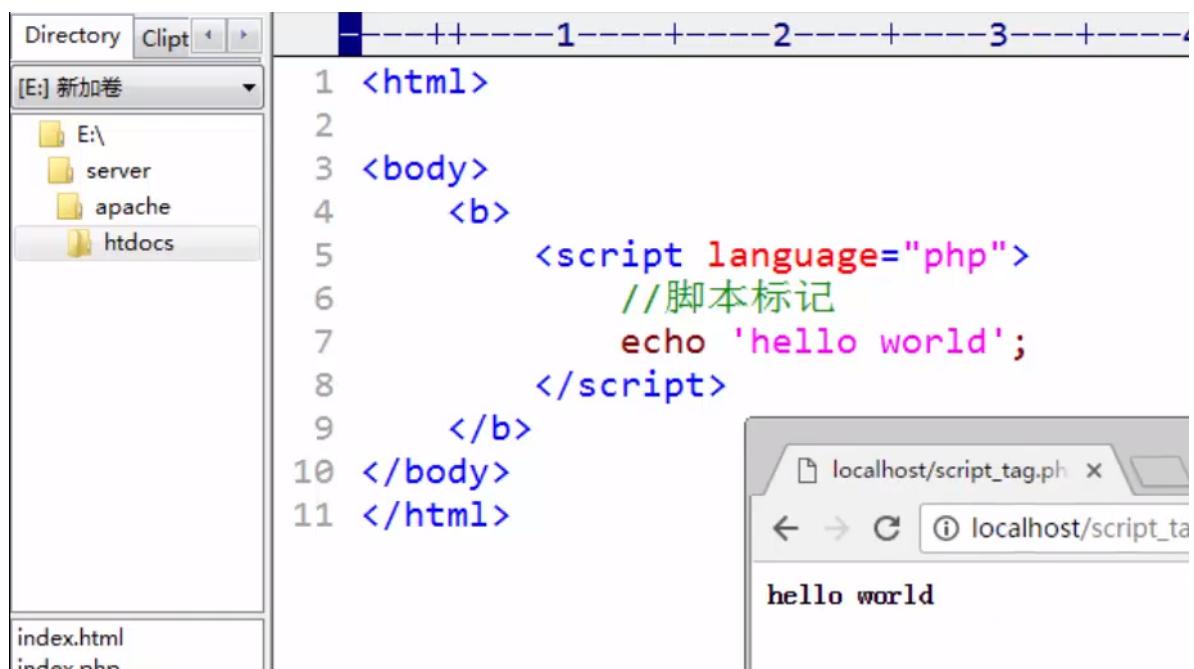
## 初步语法

PHP是一种运行在服务器端的脚本语言，可以嵌入到HTML中。（HTML是通过浏览器解析，PHP是通过PHP引擎解析，那么嵌入到里面以后如何区分什么时候通过什么来解析？）

**PHP代码标记**：可通过多种标记来区分PHP脚本

ASP标记 <% php 代码 %>； 短标记 以上两种基本弃用，如使用，需在配置文件中开启

脚本标记：也不经常用



The screenshot shows a code editor interface with a file browser on the left and a code editor on the right. The file browser shows a directory structure with 'E:\' and subfolders 'server', 'apache', and 'htdocs'. The code editor displays the following PHP script:

```
1 <html>
2
3 <body>
4   <b>
5     <script language="php">
6       //脚本标记
7       echo 'hello world';
8     </script>
9   </b>
10 </body>
11 </html>
```

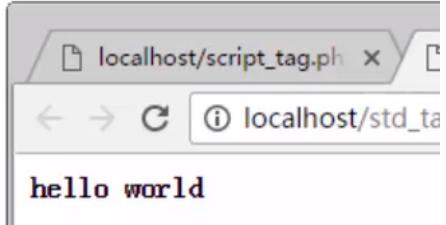
To the right of the code editor is a browser window showing the result of the script. The address bar says 'localhost/script\_tag.php'. The page content is 'hello world'.

标准标记：

```

1 <html>
2
3 <body>
4   <b>
5     <?php
6       //脚本标记
7       echo 'hello world';
8     ?>
9   </b>
10 </body>
11 </html>

```



**PHP注释**：行注释：// ( 双斜杠 ) 或# 块注释：/\* \*/

#### PHP语句分隔符：

1、在PHP中，代码以行为单位，系统需要通过判断行的结束，通常都用；表示结束。

注注注：PHP中标记结束符?>有自带语句结束符的效果，最后一行PHP代码可以没有语句结束符?>

2、PHP中代码的书写并不是全嵌入到HTML中，而是单独存在，所以可以不用标记结束符?>

## 变量

- 1、变量是用来存储数据的；
- 2、变量是存在名字的；
- 3、变量是通过名字访问数据的；
- 4、变量的数据是可以改变的。

### 变量的使用

- 1、定义：在系统中增加对应的变量名字（在内存中开发一块地）
- 2、赋值：将数据赋值给变量（在定义时直接初始化）
- 3、可通过变量名访问数据
- 4、变量可从内存中删除

```

// 定义时不需要关键字，但必须使用$符号
$var1;
$var2 = 1;
echo $var2;    // 访问变量，通过变量名找到数据，并显示
$var2 = 2;      // 修改变量
echo '<br/>', $var2; // hr/“为下划线，分隔符
unset($var2); // 删除变量，从内存中剔除
echo $var2;    // 此时会报错，因为不存在变量

```

### 变量命名规则

- 1、变量名字必须以"\$"开头；
- 2、变量名可由数字、字母、下划线命名，但必须以字母和下划线开头

3、允许中文变量；

## 预定义变量

即提前定义的变量，由系统定义的变量，存储许多要用到的数据（预定义变量都是数组）。

**\$\_GET**: 获取所有表单以 get 方式提交的数据。  
**\$\_POST**: POST 提交的数据都会保存在此。  
**\$\_REQUEST**: GET 和 POST 提交的都会保存。  
**\$GLOBALS**: PHP 中所有的全局变量。  
**\$\_SERVER**: 服务器信息。  
**\$\_SESSION**: session 会话数据。  
**\$\_COOKIE**: cookie 会话数据。  
**\$\_ENV**: 环境信息。  
**\$\_FILES**: 用户上传的文件信息。

## 可变变量

如果一个变量保存的值刚好是另外一个变量的名字，则可直接通过访问一个变量得到另外一个变量的值：但需在变量前多加一个\$符号

```
$a = 'b'; // a变量的内容正好是b变量的名称，故称a为可变变量
$b = 'bb';
echo $$a; // 使用时需加一个$符号
```

## 变量传值

将一个变量赋值给另一个变量：值传递、引用传递

值传递：将变量保存的值复制一份，并将该值给另外一个变量保存(两个变量无关系)

引用传递：将变量保存值所在的内存地址传递给另外一个变量，两个变量同指一块内存（名字不一样而已）。

## 分区

在内存中，通常有以下几个分区。

栈区：程序可以操作的内存部分（不存数据，运行程序代码），少但是快。

代码段：存储程序的内存部分（不执行）。

数据段：存储普通数据（全局区和静态区）。

堆区：存储复杂数据，大但是效率低。

注：c++中，栈区由编译器自动释放，存储局部变量和函数参数；代码区存放函数的二进制代码，由操作系统管理；全局区存放全局变量、静态变量、常量；堆区由程序员分配释放。

## 值传递

```

// 值传递
<?php

// 2.1 执行此行，在栈区开辟一块内存存储$a， 在数据段中开辟一块内存保存值1。然后将1所在位置赋值给a变量
$a = 1;

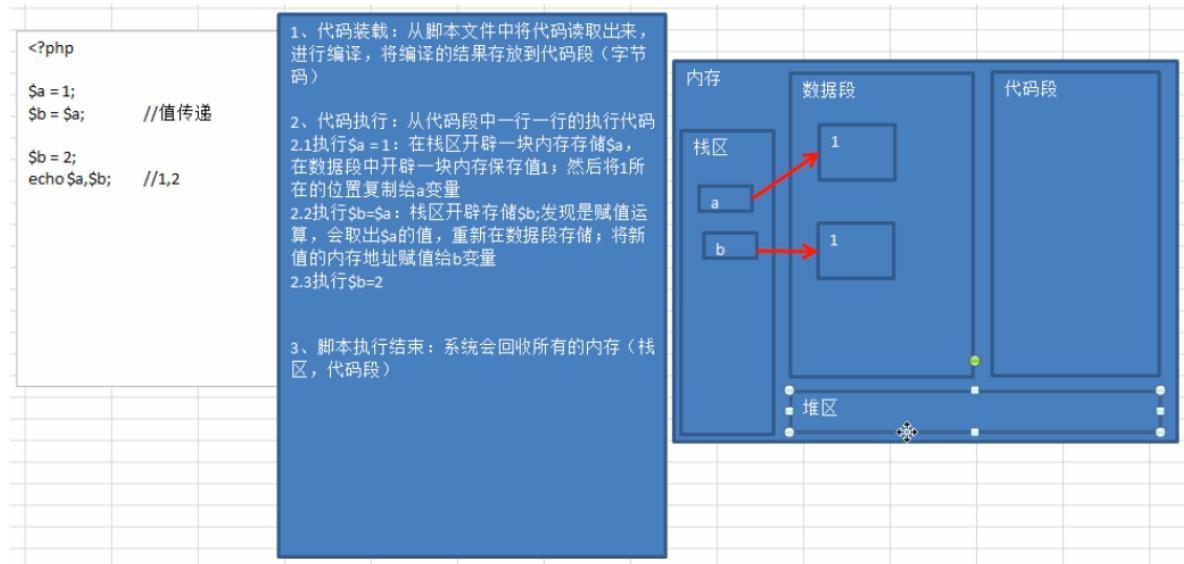
// 2.2 栈区开辟存储$b；发现是赋值运算，故会取出$a的值， 并在数据段重新开辟一块内存并保存，且再把新开辟地址赋值给栈区的变量b
$b = $a; // 值传递

// 2.3 执行该行
$b = 2;
echo $a,$b;

```

讲解运行步骤：

- 1、代码装载：从脚本文件中将代码读取出来，进行编译，将编译结果存放到代码段（二进制）。
- 2、代码执行：从代码段中一行一行执行代码。
- 3、脚本执行结束：系统会回收所有内存（栈区、代码区）：因为数据段与栈区有关系，回收栈后，数据段的内容无意义，相当于回收。



## 引用传递

```

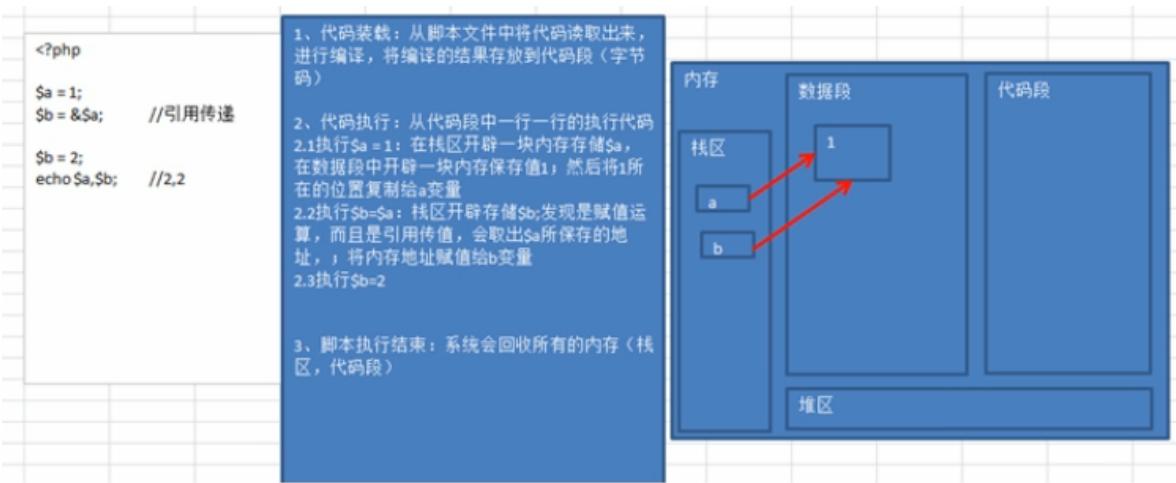
// 引用传递
<?php

// 2.1 执行此行，在栈区开辟一块内存存储$a， 在数据段中开辟一块内存保存值1。然后将1所在位置赋值给a变量
$a = 1;

// 2.2 栈区开辟存储$b；发现是引用运算，故会取出$a的地址，并将该地址给栈区的变量b
$b = &$a; // 值传递

// 2.3 执行该行
$b = 2;
echo $a,$b;

```



## 常量

常量与变量一样，均是用来保存数据的。

### 常量的基本概念

const/constant：是在程序运行中，不可改变的量（数据）；常量一旦定义，通常不可更改。

### 常量定义的形式

#### 定义方式

1、使用定义常量的函数：define ('常量名',常量值) —— 类似于c++的 #define

2、const

```
<?php
// 使用函数定义常量
define('PI',3.1415); // 注意此处与c++不同，#define 为预处理命令，宏定义，无需加；。
// 使用const关键字定义
const PI1 = 3;

// 定义特殊常量
define('^~^','smile');
// const ^-^ // 报错

// 访问常量
echo PI1;
// echo ^-^ // 报错
constant('^-^'); // 特殊常量的访问

// 系统常量
echo '<br/>',PHP_VERSION,'<br/>',PHP_INI_SIZE,'<br/>',PHP_INI_MAX; // 有符号整形

// 魔术常量
echo '<br/>__', '__DIR__', '<br/>__', '__FILE__', '<br/>__', '__LINE__';
echo __LINE__; // 输出的行数会变
```

#### 常量命名规则：

- 1、常量不需要使用"\$"符号，一旦使用被认为是变量
- 2、变量名可由数字、字母、下划线命名，但必须以字母和下划线开头（const定义）

- 3、常量的名字通常是以大写字母为主（与变量以示区别）
- 4、变量命名的规则比变量要松散，可以使用一些特殊字符（define函数）
- 5、变量通常不区分大小写，但可以区分（define函数的第三个参数）

## 系统常量

魔术常量：由双下划线+常量名+双下划线组成，其值会随着环境变化而变化，用户无法改变。

- \_DIR\_：当前被执行的脚本所在电脑的绝对路径
- \_FILE\_：当前被执行的脚本所在的电脑的绝对路径（带自己文件的名字）
- \_LINE\_：当前所属的行数
- +
- \_NAMESPACE\_：当前所属的命名空间
- \_CLASS\_：当前所属的类
- \_METHOD\_：当前所属的方法

## 数据类型

数据类型：data type，在PHP中指的是数据本身的类型，而不是变量的类型。PHP是一种弱类型语言，变量本身没有数据类型。

### PHP八大数据类型

在 PHP 中将数据分为三大类八小类：

简单（基本）数据类型：4 个小类

整型：int/integer，系统分配 4 个字节存储，表示整数类型（有前提）

浮点型：float/double，系统分配 8 个字节存储，表示小数或者整型存不下的整数

字符串型：string，系统根据实际长度分配，表示字符串（引号）

布尔类型：bool/boolean，表示布尔类型，只有两个值：true 和 false

复合数据类型：2 个小类

对象类型：object，存放对象（面向对象）

数组类型：array，存储多个数据（一次性）

特殊数据类型：2 个小类

资源类型：resource，存放资源数据（PHP 外部数据，如数据库、文件）

空类型：NULL，只有一个值就是 NULL（不能运算）

## 类型转换

很多情况需要指定数据类型，需要将外部数据类型（当前PHP获得的数据）转换成目标数据类型。

两种转化方式：

- 1、自动转化：系统根据自己的需求判断，自己转化（用的较多、但效率较低）。

2、强制转换（手动）：在变量之前增加一个()，并在括号里面写上对于的类型，其中NULL特殊，需用unset。

#### 转换说明：

- 1、以字母开头的字符串，永远为0；
- 2、以数字开头的字符串，取到碰到字符串为止。（不会同时包含两个小数点）

```
<?php
// 数据类型
// 创建数据
$a = 'abcd1.1.1';
$b = '1.1.1abc';

// 自动转换。算术运算，系统先转化为数值类型，然后运算
echo $a+$b; // 结果为1.1(0+1.1)

// 强制转换
echo '<br/>',(float)$a,(float)$b; // 01.1

// 类型判断
echo '<hr/>';
var_dump(is_int($a)); // bool(FALSE)
var_dump(is_string($a)); // bool(TRUE)

// 获取数据类型
echo '<hr/>';
echo gettype($a); // string (虽然前面强制转换了，但并未改变本身)

// 设置类型
// var_dump 输出展示展示代码内容，结构与类型。该函数作可以窥探所有内容的类型，以及内部信息
var_dump(settype($b,'int')); // 先将字符串转换为int型，转换成功返回true，var_dump判断
是否为bool型，故显示
bool(true)
echo gettype($b),$b; // interger1
```

## 整数类型

```
<?php

/*
十进制转换二进制---->除以2
10 1010 注：不管结果ruhr，均需补足32位：00000000 00000000 00000000 00001010
*/

// php中提供了很多函数进行转换：
// Decbin():十进制转二进制
var_dump(decbin(107)); // 结果：string(7) "1101011"
// 同理，还有Decoct():十进制转八进制
// Dechex():十进制转十六进制
// Bindec():二进制转十进制

*/
```

## 浮点数类型

问：为什么浮点数和整型均占用四个字节，为什么比整型表示的范围大？

整型数据的32位均通过\*2转化为十进制。而浮点型中，前八位的后七位为指数，所以表示的范围要大。

另，实际使用时，尽量不用浮点型数字做精确判断，且计算机中凡是小数基本上均不准确。

```
<?php

// 浮点数的定义
$f1 = 1.23;
$f2 = 1.23e10;
$f3 = PHP_INT_MAX+1; // 若整型超过自身存储的大小之后会自动改为浮点型存储

var_dump($f1,$f2,$f3);
// 结果: float(1.23)float(1.2300000000) float(214748348)

// 浮点数判断
$f4 = 0.7;
$f5 = 2.1;
$f6 = $f5/3;
var_dump($f6 == $f4); // 结果: bool(false),,因此其不能进行精确判断
```

## 运算符

```
<?php

// 运算符：是一种将数据进行运算的特殊符号，在PHP中一共有十多种运算符。

// 算术运算符 +-*/%
// 比较运算符 > >= < <= ==(数据大小相同即可，无需考虑数据类型) != ===(全等于，大小及数据类型均等) !==
$a = '123'; // 字符串
$b = 123; // 整型
var_dump($a == $b); // 结果: bool(true)

var_dump($a === $b); // 结果: bool(false) 不全等于

// 逻辑运算符 &&(左边条件与右边条件同时成立) ||(有一个满足即可) !(取反)
$c = 'weekend';
$d = 'goods';

var_dump($c == 'weekend' && $d == 'good'); // bool(false)
var_dump($c == 'weekend' && $d == 'good'); // bool(true)
var_dump(!( $c == good)); // bool(true)

// 连接运算符 .(将字符串连接一起) .=(将左边内容与右边内容连接起来并重新赋值)
$e = 'hello';
$f = 123;
echo $e . $f; // hello 123 (注意，此处有强制类型转换)

$e .= $e;
echo $e; // hello 123

// 错误抑制符: @ (可能出错的表达式)，在PHP中有一些错误可以提前预知，但又不想报错，这就需要错误抑制符。
$g = 0;
echo $f % $g; // 此时会报错
echo @$f % $g; // 不会报错

// 三目运算符(问号表达式) 表达式1 ? 表达式2: 表达式3
echo $g == 0 ? 1 : 2;
```

```

// 自操作运算符 ++ --(前置或后置如果只有自操作，则效果一致)
$i = $g++; // $g = 1; $i = 0
$i = ++$g; // $g = 2; $i = 2;

// 位运算符
/*
    计算机码：计算机在实际存储数据时，采用的编码规则(二进制规则)
    计算机码：原码、反码和补码。数值本身最左边一位用来充当符号位：正数为0，负数为1;
    原码：数据本身从十进制转换成二进制得到的结果
        正数，左符号位为0           负数：右符号位为1
    反码：针对负数，符号位不变，其他位取反。
    补码：针对负数，反码+1。(系统中存的为补码)
    以0为例，若是原码，则+0 = 00000000 -0 = 10000000; 二者不一样
        -0 反码 11111111 -0 补码 00000000 与+0一样（正数原码、补码、反码为其本身）。
*/
$j = 5; // 原码: 00000101
$k = -5; // 原码: 10000101 反码: 11111010 补码: 11111011

// 位运算：取出计算机中最小的单位(bit)进行运算 & | ~(按位取反) ^(按位异或) >>(右移) <<(左移)
// 注：1、系统进行位运算时，均是利用补码进行运算的      2、运算结束之后，必须转换为原码进行显示

// 按位取与
var_dump($j & $k); // int(1)
/*
    5 00000101
    -5 11111011
    & 00000001 判断：符号位为0，正数，所存即为原码，无需操作
*/
// 按位取反
var_dump(~$k); // int(4)
/*
    -5 11111011
    ~ 00000100 正数即为原码,
*/
// 按位左移
var_dump($k >> 1) // int(-3)
var_dump($k >> 2) // int(-2)
/*
    -5 11111011
    >>2 11111110(右移补符号位)
    反码 11111101(补码-1)
    原码 10000010(除符号位均取反)
*/
// 运算符优先级

```

## 流程控制

**分类：**顺序结构、分支结构（if分支与switch分支）、循环结构

<?php

```

// 分支结构—if分支
$day = '星期天';

```

```

if($day == '星期1')
{
    echo 'go out';
}
else
{
    echo 'work';
}

// switch 分支: 同一条件下, 有多个值, 且每个值对应一种操作
/*
switch(条件表达式)
{
    case 值1:
        代码;
        break;
    case 值1:
        代码;
        break;
    default:
        代码;
        break;
}
*/

// 循环结构 for循环、while循环、Do-While循环、foreach循环 (针对数组)
for($i = 0;i<10;i++)
{
    echo $i;
}

// while Do-while循环
while($i <= 10)
{
    echo $i++;
}

// 循环控制
// 1、中断控制: 重新开始从头循环 continue(需求, 输出1-100的5的倍数)
$i = 1;
while($i <= 100)
{
    if($i % 5 != 0)
    {
        $i++;
        continue;
    }
    echo $i++;
}

```

因为循环经常性会碰到嵌套（循环中间包含循环），如果在循环内部有些条件下，明确可以知道当前循环（或者说外部循环）不需要继续执行了，那么就是可以使用循环控制来实现：其中内部循环也可以控制到外部，就是通过使用层级参数。 ↴

Continue 2; //当前自己循环后面内部不再执行，同时外部循环如果还有循环体也不再执行，重新来过； ↴

Break 2; //当前自己循环结束，同时外部也结束（如果还有外部不受影响，继续执行）。 ↴

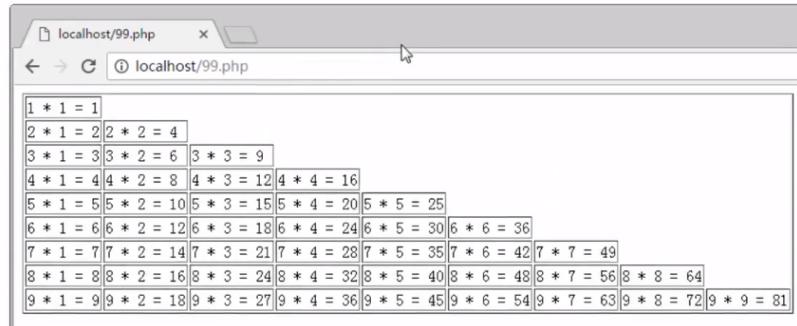
## 流程控制替代语法

流程控制替代语法：分支和循环结构的替代语法

PHP 本身是嵌入到 HTML 中的脚本语言，需要在 HTML 中书写一些关于判断或者循环的结构语法，必须符合 PHP 标签规范，需要 HTML 与 PHP 进行混搭，如果使用原始的 PHP 代码那么会非常不美观。

需求：打印一个九九乘法表，使用表格

```
1 <table border=1>
2   <?php for($i = 1;$i < 10;$i++) {?>
3     <tr>
4       <?php for($j = 1;$j <= $i;$j++) {?>
5         <td>
6           <?php echo $i . ' * ' . $j . ' = ' . $i * $j;?>
7         </td>
8       <?php }?>
9     </tr>
10    <?php }?>
11  </table>
```



上述书写中(tr 代表表格的一行，td 代表表格的一列)，大括号很容易丢失，所以 PHP 有了一种替代机制，让其可以不用书写大括号。

```
for(;;){ => : }=>endfor;

  <table border=1>
    <?php for($i = 1;$i < 10;$i++):?>
      <tr>
        <?php for($j = 1;$j <= $i;$j++) :?>
          <td>
            <?php echo $i . ' * ' . $j
          </td>
        <?php endfor?:>
      </tr>
    <?php endfor;?>
  </table>
```

PHP 中具体有哪些替代语法呢？PHP 应该在 HTML 中只做数据输出，输出通常伴有条件判断和循环操作，因此 PHP 提供了对应分支结构和循环结构的替代语法：全部都是对应的一个模式：

左大括号{使用冒号替代：

右大括号}使用 end+对应的其实标记替代

## 文件包含

文件包含：在一个 PHP 脚本中，将另外一个文件包含进来，合作完成一件事情。

### 文件包含作用

1、要么使用被包含文件中的内容，实现代码共享，向上包含（索要）

2、要么自己的东西可被使用，向下包含（给予）：自己有某个东西需要别的脚本显示。

最大的作用是分工协作，共同完成一件事情。

## 文件包含四种形式

```
<?php

// 包含文件：使用数据

// 包含文件： include  include_once(系统自动判断文件包含当中，是否已被包含过)  require
require_once
// 向上包含：使用已准备好的文件
include 'include1.php'; // 包含当前文件所在文件夹下的include1.php文件
echo $a,PI;           // include1.php中已定义这个变量和常量，故可直接使用

// 向下包含：类似于调用了子函数
$a = 10;
const PI = 3.14;
include_once 'display.php'; // 该文件中可输出a和PI
```

## 文件加载原理

### PHP代码执行流程：

- 1、读取代码文件（相当于PHP程序）；
- 2、编译：将PHP代码转化成字节码（二进制），生成opcode（php可解析的代码）；
- 3、针对引擎来解析opcode，按照细节码进行逻辑运算；
- 4 转化成对应的html代码。

1、在文件加载（include 或 require）时，系统会自动嵌入对应的include位置

2、在PHP中，被包含的文件是单独编译的。。

若编译过程中报错，则会失败，不执行。但若是被包含文件有错误，则系统执行到include语句时，才回报错。

### include 和 require区别

```

1 <?php
2
3 //被包含文件
4
5 //定义数据
6 $a = 1;
7 define('PI',3.14);
8
9 echo $a,PI;
10
11 //再次加重
12 include 'include1.php';
13

```

localhost/include2.php x localhost/include1.php x

localhost/include2.php

13.14  
Notice: Constant PI already defined in E:\server\apache\htdocs\include1.php on line 8

include会执行多次，导致报错（重复定义变量）。而include\_once不会出现这种情况。

require和include区别在于：若未包含文件，则报错形式不一样。（require包含错误文件，则include后不再执行；include未包含文件，会警告，但是仍会执行后面的。）

## 文件加载路径

文件加载时需指定文件路径，才能保证PHP正确找到对应的文件。

1、绝对路径（从磁盘根目录开始—本地绝对路径；从网站根目录开始—网络绝对路径--相对于/的路径，，/相当于绝对路径）

2、相对路径：从当前文件所在目录开始的路径。。。或者./表示当前文件夹。。。上级目录

绝对路径和相对路径加载区别：绝对路径相对效率偏低（因为要从跟目录开始找，但相对安全，路径固定）

```

<?php
// 文件加载路径
// 相对路径加载(只供演示, 不考虑多次加载)
include 'include1.php'; // 不写路径, 默认在当前文件夹下
include './include1.php'; // 另一种形式
include '../hostdoc/include1.php'; // ../代表当前文件夹的上一个文件夹, hostdoc为当前文件夹

// 绝对路径
include 'E:/server/apache/htdocs/include1.php'; // 绝对路径, 不会出错

```

## 文件嵌套包含

```

1 <?php
2 // 定义数据
3
4 $a = 10;
5 const PI = 3.14;
6
7 // 包含文件：为了显示以上数据
8 include_once 'include4.php'
9
10 // 文件嵌套包含
11 // 包含 include3.php // 文件本身包含了 include3.php
12 include 'include3.php';

```

E:\server\apache\htdocs\include4.php

```

1 <table>
2   <tr>
3     <td><?php echo $a;?></td>
4     <td><?php echo PI;?></td>
5   </tr>
6 </table>

```

localhost/include6.php

10 3.14

嵌套包含容易出现相对路径出错的问题。

## 函数

将实现某一功能的代码块封装到一个结构中，从而实现代码的复用。

### 函数定义语法（与c的差别在于可在任意位置调用子函数）

```

/*
Function 函数名(参数){
    函数体
    返回值
}
*/
// 函数的定义
// 1、函数不会自动运行，必须调用才可
// 2、代码执行阶段，遇到函数名字才回调用，不是在编译阶段
// 3、函数调用可在声明之前
function display()
{
    echo 'hello world'; // 没有返回值
}

// 函数的调用()
display(); // 若函数有参数，则需加参数

// 函数命名规范：字母数字下划线、但不能数字开头。
// 一半遵循以下规则：1、驼峰法：除第一个单词外，其余首字母大写。showParent() 2、下划线方式
// 在一个脚本函数周期中，不允许出现同名函数。

```

## 函数参数

```

<?php
// 函数参数
// 定义函数：定义函数时使用的参数，形参

```

```

function add($arg1,$arg2)
{
    echo $arg1+$arg2;
}

// 调用函数时使用的参数，实参
$num1 = 10;
add($num1,20); // 1、实参数可以多于形参（不能少于），只是函数不用而已 2、理论上实参数没有限制
/* 调用过程：1、系统调用add函数时，会去内存中找是否有add函数
   2、系统在栈区开辟内存空间运行函数add
   3、系统查看函数本身是否有形参
   4、系统判断调用函数时是否有实参
   5、系统默认会将实参$num、20分别赋值给形参
   6、执行函数体、运行
   7、返回值
*/
// 默认值：形参的默认值。。若调用时没有提供实参，则函数使用默认值执行函数
// 注：1、默认值定义时，应放在后边，不能左边有默认值，而右边没有
function moren($num1 = 0,$num2 = 0) // 当前的$num1是形参，编译时不执行。且如果外部有同名子变量，也不会冲突
{
    echo $num1-$num;
}

// 上述实参数形参的传递相当于值传递，函数内部改变变量的内容，不会影响外面变量的内容
// 引用传递：可在函数内部改变外部变量
function yinyong($a,&$b) // 函数要的是地址，故将外部变量b存储的地址取出赋值给了形参
{
    $b = $b - 1;
    $a = $a -1;
}
$a = 10;
$b = 5;
yinyong($a,$b); // 注意：此处不取地址..另，引用传递不可传入数字（常量中存储的不是地址）

```

## 函数返回值

1、return 在函数内部存在的价值：返回当前函数的结果（当前函数运行结束）

```

L3  //加法运算
L4  function add($num1,$num2){
L5      return $num1 + $num2; //return直接结束函数，所以后面所有内容都不再执行
L6
L7  //输出
L8  echo $num1;
L9 }
L10
L11 $res = add(10,20);
L12 echo $res;

```

2、return 还可以在文件中直接使用（不在函数里面）：代表文件将结果 return 后面跟的内容，转交给包含当前文件的位置。（通常在系统配置文件中使用较多）

## 作用域（与c差别在于全局变量不能直接被函数调用）

作用域：通常是指变量可以被访问的区域。

在PHP中，作用域严格分为两种，以及内部定义的一种

1、全局变量：所属全局空间，在PHP中只允许在全局空间使用，函数内部不可用。（c++可使用）

## 2、局部变量：函数内部的变量

3、超全局变量：预定义变量（系统定义的），没有访问限制，能够帮助局部去访问全局变量。

```
<? php

// php中作用域

// 默认的代码空间：全局空间
$global = 'global area';

// 局部变量（函数内部定义）
function display()
{
    $inner = 1;

    // 访问全局变量
    echo $global; // 函数内部不能访问全局变量

    // 转化为超全局变量，使得函数内部可以访问
    echo $GLOBALS['global']; // 这样可访问
}

display();
```

想在函数内部访问全局变量，可通过\$GLOBALS，也可使用引用传值。

另，还有一种方式既可从全局访问局部、也可从局部访问全局。即，global关键字：

1、若使用global定义的关键字在外部存在，那么系统在函数内部定义的变量直接指向外部变量所指向的内存空间（同一个变量）。

2、若其定义的变量在外部不存在，系统会自动在全局空间定义一个与局部变量同名的全局变量。

本质为：在函数的内部和外部，对一个同名变量使用同一块内存地址保存数据。

```
<? php

// global关键字的应用

// 默认的代码空间：全局空间
$global = 'global area';

// 局部变量（函数内部定义）
function display()
{
    // 访问全局变量
    echo $global; // 函数内部不能访问全局变量

    // 1、全局变量存在
    global $global;
    echo $global; // 此时可以调用全局变量
    //2、全局变量不存在
    global $local = 'inner';
}

echo $local; // 访问局部变量
display();
```

## 静态变量

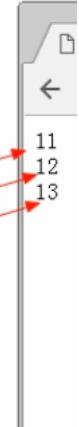
静态变量：static是在函数内部定义的变量，使用static关键字修饰，用来实现跨函数共享数据的变量（注：跨函数是指同个函数多次调用）。

```
// 定义函数
function display(){
    // 定义变量
    $local = 1; // 局部变量

    // 定义静态变量
    static $count = 1; // 静态变量

    echo $local++, $count++, '<br/>';
}

// 调用
display();
display(); // 第二次调用时，$local=2, $count=2
display(); // 第三次调用时，$local=3, $count=3
```



静态变量原理：系统在进行编译的时候，就已经对static这一行进行了初始化：即为静态变量赋值。。。而在执行时，会跳过初始化这一行。

## 可变函数（未写全）

```
<?php

// 可变函数：当前有一个变量所保存的值，刚好是一个函数的名字，那么就可以使用$变量()来充当函数名使用。
// 可变函数在使用系统函数时，需用户在外部定义一个自定义函数，但是需要传入到系统函数内部使用。
// 定义函数
function display()
{
    echo __FUNCTION__;
}

// 定义变量
$func = 'display';

// 可变函数
$func();
```

## 匿名函数

```
<?php

// 匿名函数：没有名字的函数，

// 定义基本匿名函数
// function() // 没有办法运行，故须加一个变量名字
// 变量保存匿名函数，本质得到的是一个对象（closure类中的对象）
$func = function()
{
    echo 'hello world';
}; // 因为相当于变量的赋值，所以需加一个分号
```

```

// 调用匿名函数
$func();

/*
闭包：要执行的代码块（由于自由变量被包含在代码块中，这些自由变量以及它们引用的对象没有被释放）和为自由变量提供绑定地计算环境。（简单理解就是说，函数内部的一些局部变量即要执行的代码块，在执行完毕后没有被释放）。没有被释放的原因是：在函数内部，还有对应的函数被引用，通常为匿名函数。
*/



// 闭包函数
function display()
{
    $name == __FUNCTION__;

    // 定义匿名函数
    // $innerfunction = function()// $name 相对于匿名函数来说是外部变量，故不能直接用
    // 1、使用匿名函数 2、使用关键字use
    $innerfunction = function() use($name) // use就是将局部变量 保留给内部使用（形成了闭
包，使display运行完
    {
        echo $name;
    }
    $innerfunction();
    // 3、匿名函数返回给外部使用
    return $innerfunction(); // 为验证局部变量未被释放而返回
};

$closure = display(); // 理论上此处局部变量被释放

$closure(); // 结果却输出了name，说明上一行并未释放局部变量

// 如何证明局部变量在函数使用完之后没有被释放？（三步法）
// 1、使用内部匿名函数
// 2、匿名函数使用句变量：use
// 3、匿名函数返回给外部使用。

```

## 伪类型

PHP中不存在的类型，只是为了方便查看操作手册。（可当作模板，只是为了泛化数据类型）

伪类型有两类:mixed混合的，只要是php中的类型即可；numbe数值的

## 常用系统函数

```

<?php

// 系统函数

// 输出相关
// print():类似于echo输出提供的内容，本质是一种结构（非函数），返回值为1
echo print('hello lewao'); // 输出hello world
print 'hello lewao'; // 输出1hello world (1为上一行echo的输出)

// print_r(): 类似于var_dump，但不会输出数据类型，只会输出值，数组打印使用较多。
print_r('hello'); // 输出为hello

//时间函数
echo date('Y 年 m 月 d 日');
echo time();
echo microtime();

```

```

// 数学函数： max()、rand():指定范围里的随机整数、round(): 四舍五入、ceil():向上取整、
floor():向下取整、pow(2,8):2的8次方、abs(): 绝对值、sqrt(): 求平方根

// 有关函数的函数
function test($a,$b)
{
    // 获取指定参数
    var_dump(func_get_arg(1));      // 得到了第二个参数，结果为：string(1) '2';
    // 获取所有参数:对应实参的个数
    var_dump(func_get_args());     // array(4){    }
    // 获取参数数量: 对应实参的个数
    var_dump(func_num_args());     // 参数数量 int(4)
}
// 调用(如果存在的话，执行test)
function_exist('test') && test(1,'2',3,4);

```

## 错误处理

错误处理：指系统或用户在对某些代码进行执行的时候，发现有错误，就会通过错误处理的形式告诉程序员。

### 错误分类

- 1、语法错误：书写代码不符合PHP语法规则，会导致代码在编译中不允许，故也不会执行（parse error）；
- 2、运行时错误：代码编译通过，但在执行时会出现一些条件不满足从而导致的错误。（runtime error取空数组的第几位数）
- 3、逻辑错误：写代码不规范、但逻辑性错误，导致虽可正常运行，但得不到预期结果。

### 错误代号

系统代号在PHP中均被定义为了系统常量，故可直接使用：

- 1、系统错误（系统使用的代号）：

E\_PARSE：编译错误，代码不会运行

E\_ERROR：fatal error致命错误，会在出错的位置断掉

E\_WARNING：warning警告错误，不影响执行，但可能得不到预期结果

E\_NOTICE：notice，通知错误、不影响代码执行

- 2、用户错误（用户使用的代号）：E\_USER\_ERROR、E\_USER\_WARNING、E\_USER\_NOTICE用户在使用自定义错误出发的时候，会使用道德错误代号。

- 3、E\_ALL：代表所有错误

所有E开头的错误常量都由一个字节（8位）存储，且每一种错误占用一个位，故可进行位操作。

排除通知级别notice：E\_ALL & ~E\_NOTICE。。假设ALL全为1，那么与NOTICE取反再取与就可将其剔除

只要警告和同志：E\_WARNING | E\_NOTICE

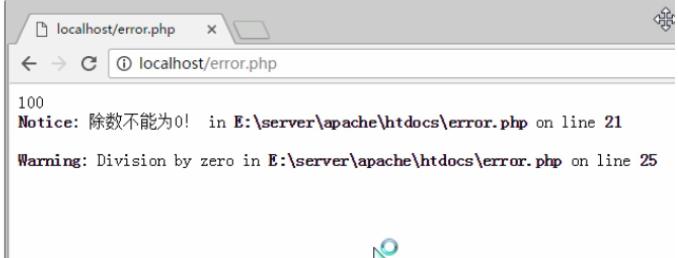
## 错误触发

程序运行时触发：主要针对代码的语法错误和运行时错误。

人为触发：知道某些逻辑可能会出错，从而使用对应的代码编号来判断

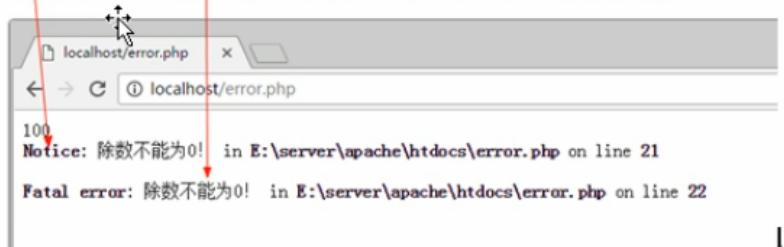
```
<?php  
  
// php错误处理  
  
// 人为触发  
// 处理脚本让浏览器按照指定字符集解析  
// header('Content-type:text/html;charset=utf-8');  
  
$b = 0;  
if($b == 0){  
{  
    trigger_error('除数不能为0')  
}  
echo $a / $b;
```

```
//输出  
echo $a;  
  
//除法运算  
$b = 0;  
  
if($b == 0){  
    //人为触发错误  
    trigger_error('除数不能为0! ');  
}  
  
//
```



可以通过第二个参数进行严格性控制。

```
//除法运算  
$b = 0;  
  
if($b == 0){  
    //人为触发错误  
    trigger_error('除数不能为0! '); //默认notice, 会继续执行  
    trigger_error('除数不能为0!', E_USER_ERROR); //默认error, 代码不会执行  
}  
  
echo 'hello ';
```



## 错误显示设置（未全）

## 字符串类型

### 字符串定义语法

```
<?php
```

```

//php字符串：定义

// 1、引号定义：比较适合定义较短的或无结构要求的字符串
$str1 = 'hello';
$str2 = "hello";
var_dump($str1,$str2); // 两种方式显示的结果一致

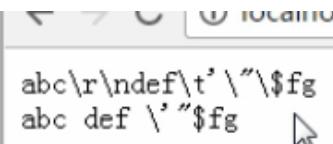
// 结构化定义
// 2、heredoc字符串：没有单引号的单引号字符串
$str3 = <<<EOD
    HELLO
    EOD;
// 3、nowdoc 结构
$str4 = <<<'EOD' // eod只是边界符，可自己定义
        hello
EOD;
var_dump($str3,$str4);

```

## 转义字符串

字符串转义：在计算机通用协议中，有一些特定方式定义的字母，系统会特定处理；反斜杠+字母  
':在单引号字符串中显示单引号 " 在双引号字符串中显示双引号 \r：回车 \n：换行 \t:四个空格 \$  
**区别：**上述转义符中，单引号只能识别。而双引号中不能识别；

```
//var_dump($str3,$str4);
```



```
//定义字符串识别转义符号
```

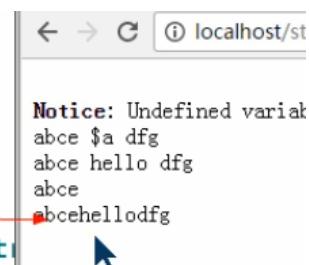
```
$str1 = 'abc\r\ndef\t\'\"\\$fg';
$str2 = "abc\r\ndef\t\'\"\\$fg";
```

```
echo $str1,'<br/>', $str2;
```

双引号中变量识别规则：

- 1、变量本身系统能够与后面的内容区分；即应该保证变量的独立性，不要使系统难以识别。
- 2、使用变量专业标识符，即给变量加一组{}；

```
$a = 'hello';
//变量识别
$str1 = 'abce $a dfg';
$str2 = "abce $a dfg";
$str3 = "abce$adfg";
$str4 = "abce{$a}dfg";
echo $str1,'<br/>', $str2,'<br/>', $str3,'<br/>', $str4,'<br/>';
```



结构化字符串变量的规则

- 1、结构化定义字符串对应的边界符有条件

- 1.1 上边界符和后面不能有任何东西（包括注释）

## 1.2 下边界符必须顶格

### 1.3 下边界符后面只能跟分号

2、结构化定义字符串内部均是字符串本身（包括空格、换行、注释等）

```
1 echo $str1,'<br/>', $str2, '<br/>', $st
2
3 abce $a dfg<br/>abc hello dfg<br/>abce<br/>abcehellodfg //这是弹出内容
4 <script>
5   alert(' abce $a dfg'); //js弹出字符串必须要有引号
6 </script>
7
8 EOD;
9
10 echo $str1;
```

## 字符串长度问题

```
<?php

header('Content-type:text/html;charset = utf-8');

// 定义字符串
$str1 = 'abcefjdoifaoi';
$str2 = '你好中国123';

echo strlen($str1), '<br/>', strlen($str2); // 13 15(中文在utf下占3个字节)

// 多字节字符串的长度问题：包含中文的长度
// 多字节字符串扩展模块：mbstring扩展(mb:Multi Bytes)
// 首先需加载PHP的mbstring扩展（php.ini中去注释即可）
// 使用mbstring
echo mb_strlen($str1), '<br/>', mb_strlen($str2); // 13 15(与之前一致)

// 长度并未改变，MBstring针对不同的字符集有不同的统计结果
echo mb_strlen($str1), '<br/>', mb_strlen($str2), '<br/>', mb_string($str2, 'utf-8'); // 13 15 7
```

## 字符串相关函数

1) 转换函数：implode(), explode(), str\_split()

Implode(连接方式,数组): 将数组中的元素按照某个规则连接成一个字符串

Explode(分割字符,目标字符串): 将字符串按照某个格式进行分割，变成数组

中国|北京|顺义 == array('中国','北京','顺义');

Str\_split(字符串,字符长度): 按照指定长度拆分字符串得到数组

↓

2) 截取函数：trim(), ltrim(), rtrim()

Trim(字符串[,指定字符]): 本身默认是用来去除两边的空格（中间不行），但是也可以指定要去除的内容，是按照指定的内容循环去除两边有的内容：直到碰到一个不是目标字符为止

Ltrim(): 去除左边的

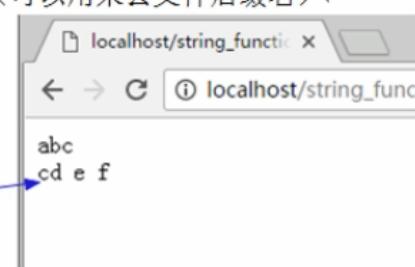
Rtrim(): 去除右边的

### 3) 截取函数: substr(), strstr()

Substr(字符串,起始位置从 0 开始[长度]): 指定位置开始截取字符串, 可以截取指定长度(不指定到最后)

strstr(字符串,匹配字符): 从指定位置开始, 截取到最后(可以用来去文件后缀名)

```
$str = 'abcd e f';
//字符串截取
echo substr($str,1,3),'  
>';
echo strstr($str,'c');
```



### 4) 大小转换函数: strtolower(), strtoupper(), ucfirst()

Strtolower: 全部小写

Strtoupper: 全部大写

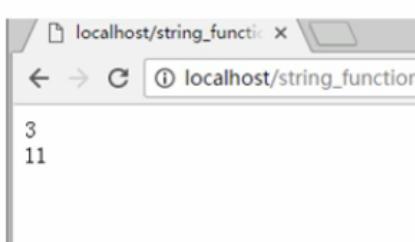
Ucfirst: 首字母大写

### 5) 查找函数: strpos(), strrpos()

Strpos(): 判断字符在目标字符串中出现的位置(首次)

Strrpos(): 判断字符在目标字符串中最后出现的位置

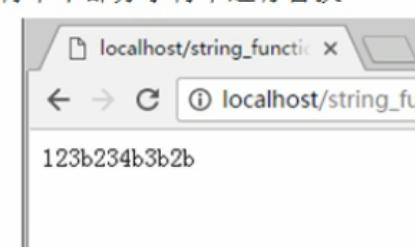
```
$str = '123a234a3b2a';
//查找位置
echo strpos($str,'a'),'  
>';
echo strrpos($str,'a');
```



### 6) 替换函数: str\_replace()

Str\_replace(匹配目标,替换的内容,字符串本身): 将目标字符串中部分字符串进行替换

```
$str = '123a234a3b2a';
//字符串替换
echo str_replace('a','b',$str);
```



### 7) 格式化函数: printf(), sprintf()

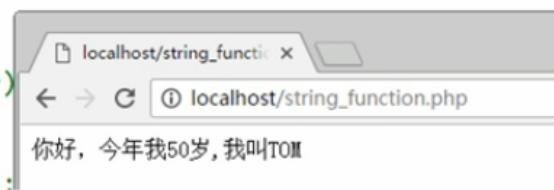
Printf/sprintf(输出字符串有占位符,顺序占位内容..): 格式化输出数据

```
$age = 50;
$name = 'TOM';

//格式化输出
echo sprintf("你好, 今年我%d岁,我叫%s",$age,$name);

//字符串替换
//echo str_replace('a','b',$str)

//查找位置
//echo strpos($str,'a').'  
>';
```



- 8) 其他: `str_repeat()`, `str_shuffle()`  
`Str_repeat()`: 重复某个字符串 N 次  
`Str_shuffle()`: 随机打乱字符串

```
//其他字符串函数
$str = 'abcdefg';

echo str_repeat($str,5), '<br/>';
echo str_shuffle($str);
```

## 数组 (元素下标均和C++不一样)

数组 : 数据的组合 , 指将一组数据 (多个) 存储到一个指定的容器中 , 并用变量指向该容器 , 然后可用变量一次性取得该容器的所有数据。

### 定义语法

```
<?php

// php数组: 可以是一种或多种类型的数据, 这与C++很不一样. 类似于哈希表

// 定义数组: array
$arr1 = array('1', 2, 'hello');
var_dump($arr1); // 结果: array(3) ([0]=>string(1) "1" [1]=>int(2)
[2]=>string(5) "hello")

// 定义数组: []
$arr2 = ['1', 2, 'hello'];
var_dump($arr2); // array(3) ([0]=>string(1) "1" [1]=>int(2) [2]=>string(5) "hello")

// 定义数组: 隐型数组
$arr3[] = 1; // 默认给数组第0个元素赋值
$arr3[10] = 100; // 第10个元素赋值
$arr3[] = '1'; // 第11个。默认下标是从当前最大下标
$arr['key'] = 'key'; // 第key个
$arr[1] = 'value' // 第1个, 但不会自动调整, 还是会处于最后一个位置
// 结果为: array(4) ([0]=>int(1) [10]=>int(100) [11]=>string(1) "1"
["key"]=>string(3)"key" [1]=>string(5) "value")
```

### PHP数组特点

1、可以整数下标或者字符串下标

若数组下标均为整数 , 则称为**索引数组**

若数组下标均为字符串 , 则称为**关联数组**。

混合下标的话称为**混合数组**

2、数组元素的顺序以放入顺序为准 , 与下标无关

3、数字下标的增长特性 : 从0开始自动增长 , 若中间手动加入较大的下标 , 则后面则会从当前最大下标 +1 增长。

4、特殊值下标的自动转换

```

<?php

// 特殊下标自动转换
$arr1[false] = false;
$arr1[true] = true;
$arr1[NULL] = NULL;
var_dump($arr1); // array(3){[0]=>bool(false), [1]=>bool(true) [""]->NULL)

```

5、PHP数组中类型元素没有限制。

6、PHP中数组元素没有长度限制。c++ vector

补充：PHP中数组是很大的数据，故会存储在堆区。

## 多维数组

二维数组：数组中所有的元素都是一维数组。

```

// 多维数组

// 定义二维数组
$info = array(
    array('name' => 'Jim', 'age' => 30),
    array('name' => 'Tom', 'age' => 28),
    array('name' => 'Lily', 'age' => 20)
    // 最后一个元素，后面可以跟逗号不影响（不建
);
echo '<pre>';
print_r($info);

```

```

Array
(
    [0] => Array
        (
            [name] => Jim
            [age] => 30
        )
    [1] => Array
        (
            [name] => Tom
            [age] => 28
        )
    [2] => Array
        (
            [name] => Lily
            [age] => 20
        )
)

```

在第二维的数组元素中可以继续是数组，在 PHP 中没有维度限制（PHP 本质并没有二维数组）。

但是：不建议使用超过三维以上的数组，会增加访问的复杂度，降低访问效率。

## · 异形数组（不规则数组）

异形数组：数组中的元素不规则，有普通基本变量也有数组。

在实际开发中，并不常用，尽量让数组元素规则化（便于进行访问）。

## 数组遍历

数组遍历：普通数组数据的访问都是通过数组元素的下标来实现访问，如果说数组中所有的数据都需要依次输出出来，就需要我们使用到一些简化的规则来实现自动获取下标以及输出数组元素。

```

$arr = array(0=>array('name' => 'Tom'), 1=>array('name' => 'Jim')); // 二维数组
// 访问一维元素： $arr[一维下标]
$arr[0]; // 结果： array('name' => 'Tom');
// 访问二维元素： $arr[一维下标][二维下标]
$arr[1]['name']; // Jim

```

## Foreach语法

```

<?php

// 数组遍历 foreach
$arr = array(1,2,3,4,5,6,7,8,9,10);

```

```

// foreach
foreach($arr as $a)
{
    echo $a,'<br/>';      // 依次输出1, 2, 3, 4
}

foreach($arr as $a => $v)
{
    echo 'key',$a,'== value',$v,'<br/>';      // 依次输出key0 == value1 等
}

// 二维数组
$arr = array(
    0 => array('name' =>'Tom','age' => 10),
    1 => array('name' => 'Jim','age' => 11)
);

// 通过foreach遍历二维元素
foreach($arr as $a)
{
    echo 'name is:',$a['name'],'age is:',$a['age'],'<br/>';
    // name is:TOM age is:10
    // name is:TOM age is:10
}

```

## foreach遍历原理

Foreach遍历的原理：本质是数组的内部有一颗指针，默认是指向数组元素的第一个元素，foreach就是利用指针去获取数据，同时移动指针。 ↻

```

Foreach($arr as $k => $v){  
    //循环体  
}  


```

- 1、 foreach会重置指针：让指针指向第一个元素； ↻
- 2、 进入 foreach循环：通过指针取得当前第一个元素，然后将下标取出放到对应的下标变量\$k中（如果存在），将值取出来放到对应的值变量\$v中；（指针下移） ↻
- 3、 进入到循环内部（循环体），开始执行； ↻
- 4、 重复 2 和 3，直到在 2 的时候遇到指针取不到内容（指针指向数组最后） ↻

## for循环遍历

```

<?php  
    // for循环遍历数组  
  
    // 数组特点：1、索引数组    2、下标规律  
    $arr = array(1,2,3,4,5,6,7,8);  
    for($i = 0; $i < count($arr); $i++)  
    {  
        echo 'key is:',$i,'value is:',$arr[$i];  
    }

```

## while配合each和list遍历数组

**While** 是在外部定义边界条件，如果要实现可以和 **for** 循环。 ↴

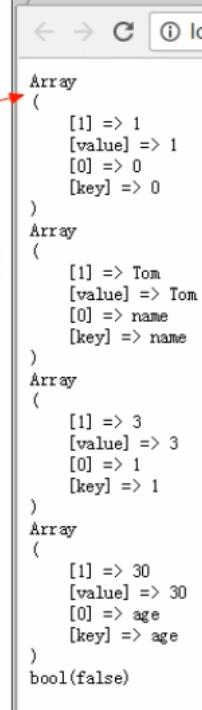
↳

**Each** 函数使用： **each** 能够从一个数组中获取当前数组指针所指向的元素的下标和值，拿到之后将数组指针下移，同时将拿到的元素下标和值以一个四个元素的数组返回： ↴

- 0 下标 - ↳ 取得元素的下标值 ↴
- 1 下标 - ↳ 取得元素的值 ↴
- Key 下标 - ↳ 取得元素的下标值 ↴
- Value 下标 - ↳ 取得元素的值 ↴

//while配合each和list遍历数组

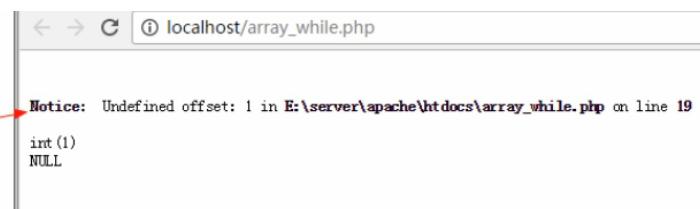
```
$arr = array(1, 'name' => 'Tom', 3, 'age' => 30);  
  
echo '<pre>';  
//each函数指针操作  
print_r(each($arr));  
print_r(each($arr));  
print_r(each($arr));  
print_r(each($arr));  
var_dump(each($arr));
```



```
Array  
(  
    [1] => 1  
    [value] => 1  
    [0] => 0  
    [key] => 0  
)  
Array  
(  
    [1] => Tom  
    [value] => Tom  
    [0] => name  
    [key] => name  
)  
Array  
(  
    [1] => 3  
    [value] => 3  
    [0] => 1  
    [key] => 1  
)  
Array  
(  
    [1] => 30  
    [value] => 30  
    [0] => age  
    [key] => age  
)  
bool(false)
```

**List** 函数使用： **list** 是一种结构，不上一种函数（没有返回值），是 **list** 提供一堆变量去从一个数组中取得元素值，然后依次存放到对应的变量当中（批量为变量赋值：值来源于数组）：  
**list** 必须从索引数组中去获取数据，而且必须从 0 开始。 ↴

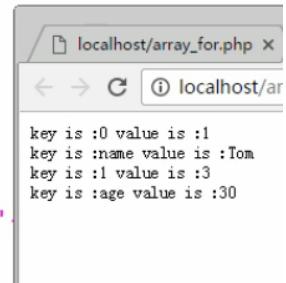
```
//list结构  
$arr = array(1, 2 => 1);  
  
list($first) = $arr;  
list($first,$second) = $arr;  
var_dump($first,$second);
```



```
Notice: Undefined offset: 1 in E:\server\apache\htdocs\array_while.php on line 19  
int(1)  
NULL
```

由于不存在下标1，而list只能从0开始依次查找，故上述会报错。

```
//while循环  
$arr = array(1, 'name' => 'Tom', 3, 'age' => 30);  
  
while(list($key,$value) = each($arr)){  
    //list搭配each  
    //list($key,$value) = each($arr);  
  
    //输出  
    echo 'key is : ' . $key . ' value is : ' . $value . '  
}
```



```
key is :0 value is :1  
key is :name value is :Tom  
key is :1 value is :3  
key is :age value is :30
```

## 数组相关的函数

<?php

```

// 数组相关函数

// 排序函数
$arr = array(3,1,5,2,0);

print_r(sort($arr)) // 结果为1
print_r($arr);      // 排序后，索引变为01234

// 指针函数 reset:将数组的内部指针指向第一个单元; end():将数组指针指向最后一个元素;
// next():    prev():指针上移    current():获取当前指针对应的元素值    key()获取当前指针对应的key

// 判断是否指针移动
echo current($arr), '<br/>';
echo key($arr), '<br/>'; //若是第一个元素。则当前数组指针未移动

echo next($arr),next($arr), '<br/>'; //15
echo prev($arr); //1
// 注意事项: next, prev会移动指针, 可能导致超出数组, 此时再使用next、prev便不能再返回数组, 只能通过end/reset

// 其他函数: count

```

```

//栈: 先压栈后出栈: 都是从一端出来
//前面: array_shift/array_unshift
//后面: array_push/array_pop
//压栈
array_push($arr,3);
array_push($arr,2);
array_push($arr,1);
print_r($arr);
//出栈
echo array_pop($arr),array_pop($arr),ar

```

```

Array
(
    [0] => 3
    [1] => 2
    [2] => 1
)
123

```

```

//队列: 先排队, 先出来, 一端进, 另外一端出
//后进前出: array_push/array_shift
//前进后出: array_unshift/array_pop
$arr = array();

//入队
array_unshift($arr,3);
array_unshift($arr,2);
array_unshift($arr,1);
print_r($arr);
//出队
echo array_pop($arr),array_pop($arr),array_pop($arr), '<br/>';

```

```

Array
(
    [0] => 1
    [1] => 2
    [2] => 3
)
321

```

```

$arr = array(1,2,3,6,5);
print_r(array_reverse($arr));

```

```

Array
(
    [0] => 5
    [1] => 6
    [2] => 3
    [3] => 2
    [4] => 1
)

```

# 编程思想

## 递推算法

利用特定关系得出中间推论，直至得到结果的算法。。。分为顺推和逆推两种

顺推：通过最简单的条件，逐步推演结果。

逆推，通过结果找到规律，从而推到已知条件。

## 递归算法

把问题转化为规模缩小了的同类问题的子问题。然后递归调用函数以表达问题的解。

**简化问题，找到最优子问题。**

递归的本质是函数调用：一个函数需要开辟一块内存，递归会出现同时调用多个函数，故占用很多内存。

## 冒泡排序

- 1、比较相邻元素，若前一个比后一个大，则交换。
- 2、对每一对相邻元素进行1操作，直至最后一对。此时最后一个因为最大值。
- 3、除最后一个外，重复以上操作
- 4、重复以上操作，直至排序完成。

```
<?php

$arr = array(1,3,2,5,9,8,7);
// 计算长度
$len = count($arr);

for($i = 0;$i<$len-1;$i++) // 第几次重复循环
{
    for($j = 0;$j<$len-$i-1;$j++) // 第几次交换
    {
        if($arr[$j] > $arr[$j+1])
            swap1($arr[$j],$arr[$j+1]);
    }
}
var_dump($arr);

// 交换值
function swap1(&$a,&$b)
{
    $tmp = $a;
    $a = $b;
    $b = $tmp;
}
```

## 选择排序

- 1、假设第一个元素为最小元素，记下下标
- 2、寻找右侧剩余元素，若有更小的，则记下更小的下标
- 3、一行对比完成后，交换第一个和最小的元素

#### 4、重新开始以上操作

```
<?php

$arr = array(1,5,2,5,9,6,3,4);
$min_index = 0;

$len = count($arr);

for($i = 0; $i<$len-1; $i++)
{
    $min_index = $i;
    for($j = $min_index + 1;$j<$len;$j++)
    {
        if($arr[$min_index] > $arr[$j])
            $min_index = $j;
    }
    if($min_index != $i)
        swap1($arr[$min_index],$arr[$i]);
}
var_dump($arr);

// 交换值
function swap1(&$a,&$b)
{
    $tmp = $a;
    $a = $b;
    $b = $tmp;
}
```

## 插入排序

- 1、认定一个第一个元素已经排好序；
- 2、取出第二个元素作为待插入元素；
- 3、将待插入元素与已排好元素比较；
- 4、若小于已排好元素，则说明前面排序未在正确位置，应该向后移动，让新元素插入进去
- 5、重复以上操作，直到该元素插入完毕
- 6、重复操作，直至所有元素完毕

```
<?php
$arr = [4,2,6,8,9,5];

$len = count($arr);

for($i = 1;$i<$len;$i++) // 第几个元素为待插入元素
{
    $tmp = $arr[$i];
    for($j = $i;$j>0;$j--) // 比较几次
    {
        if($tmp < $arr[$j-1]) // 注意此处比较的是$tmp
            $arr[$j] = $arr[$j-1];
        else
            break;
    }
    if($arr[$j] != $tmp)
```

```

        $arr[$j] = $tmp;
    }

var_dump($arr);

// 交换值
function swap1(&$a,&$b)
{
    $tmp = $a;
    $a = $b;
    $b = $tmp;
}

```

## 快速排序法：

```

<?php

$arr = array(1,6,3,4,9,2);

// 定义数组开头及结尾
$start = 0;
$end = count($arr)-1;

// 定义函数作为递归函数
function quick_sort($arr)
{
    // 递归出口
    $len = count($arr);
    if($len <= 1) return $arr;

    // 比较并分散数据
    $left = $right = array(); // 定义空数组用于存放大的或小的
    for($i = 1;$i<$len;$i++)
    {
        if($arr[$i]>$arr[0]) $right[] = $arr[$i]; // 大于存放在右边数组
        else $left[] = $arr[$i]; // 小于存放在左边数组
    }

    // 递归
    $left = quick_sort($left); // 分别将右数组和左数组进一步排序
    $right = quick_sort($right);

    return array_merge($left,(array)$arr[0],$right);
}

$res = quick_sort($arr);
var_dump($res); // 不能用echo

```

```

<?php

// 快排双指针

$arr = array(1,2,3,9,2,8,7);
function quick_sort($arr)
{
    // 递归结束条件
    $len = count($arr);
    if($len <= 1) return $arr;
}

```

```

// 定义双指针
$left = 0;
$right = $len-1;
$pivot = $arr[0]; // 确定基准
while($left < $right) // left == right时跳出循环
{
    while($left < $right && $arr[$right] >= $pivot) $right--;
    while($left < $right && $arr[$left] <= $pivot) $left++;
    swap1($arr[$left],$arr[$right]);
}
swap1($arr[$left],$arr[0]); // 跳出循环时, left = right , 故将pivot赋值即可

// 递归点(上述操作完毕后, 需要进一步排序左边和右边, 故需递归)
$left_arr = quick_sort(array_slice($arr,0,$left));
$right_arr = quick_sort(array_slice($arr,$left+1));

return array_merge($left_arr,(array)$arr[$left],$right_arr);
}

var_dump(quick_sort($arr));

```

// 交换值

```

function swap1(&$a,&$b)
{
    $tmp = $a;
    $a = $b;
    $b = $tmp;
}

```

## 归并排序

```

<?php

// 二路合并算法
$arr1 = array(1,2,5,6);
$arr2 = array(3,7,8,9);

$res = array(); // 存合并后的元素

while(count($arr1) && count($arr2))
{
    $res[] = $arr1[0]>$arr2[0] ? array_shift($arr2):array_shift($arr1);
}
while(count($arr1) && !count($arr2))
{
    $res[] = array_shift($arr1);
}
while(count($arr2) && !count($arr1))
{
    $res[] = array_shift($arr2);
}

var_dump($res);

```

1、将数组拆分成两个数组

2、重复步骤1，将数组拆分成最小单元

3、然后二路归并

4、重复步骤直至完成

```
<?php

// 归并排序

$arr = array(1,15,7,9,8,6,2,3,4);

function merge_sort($arr)
{
    // 1、递归结束条件
    $len = count($arr);
    if($len <= 1) return $arr;

    // 2、重复的操作
    // 2、1 拆分数组
    $middle = $len>>1;
    $left = array_slice($arr,0,$middle);
    $right = array_slice($arr,$middle);

    // 4、递归点（最后写这一步，先将一层的分离和合并写完，再找递归点）
    $left = merge_sort($left);
    $right = merge_sort($right);
    // 2、2 二路归并
    $res = array();      // 存合并后的元素

    while(count($left) && count($right))
    {
        $res[] = $left[0]>$right[0] ? array_shift($right):array_shift($left);
    }
    while(count($left) && !count($right))
    {
        $res[] = array_shift($left);
    }
    while(count($right) && !count($left))
    {
        $res[] = array_shift($right);
    }

    // 3、返回值
    return array_merge($res);
}

var_dump(merge_sort($arr));
```

## 查找算法

```
<?php

$arr = array(1,3,6,17,24,31,32);
// 顺序查找：从数组第一个元素开始挨个匹配
function find_index($arr,$num)
{
    foreach($arr as $k => $v)
    {
        if($v == $num) return $k;
```

```
    }

    return false;
}

// var_dump(find_index($arr,2)); // 不存在, 故返回bool(false)
// var_dump(find_index($arr,32)); // 存在, 故返回int(5)

function Binary_search($arr,$num)
{
    $len = count($arr);
    $left = 0;
    $right = $len-1;

    while($left <= $right)
    {
        $middle = $left+($right-$left>>1);
        if($arr[$middle] > $num)
            $right = $middle-1;
        elseif($arr[$middle] < $num)
            $left = $middle+1;
        else
            return $middle;
    }

    return false;
}
print_r(Binary_search($arr,6));
var_dump(Binary_search($arr,2));
```