一、PHP 函数的定义和应用

a) 函数是什么

i. 函数是一段完成指定任务的已命名代码块 ,函数可以遵照给它的一组值或参数完成特定的任务，并且可以返回一个值。在 PHP 中有两种函数：自定义函数和系统函数

ii. 函数就是按照功能的进行区分，每个一个功能定义成一个函数

b) 函数的优越性

i. 控制程序设计的复杂性

ii. 提高软件的可靠性

iii. 提供软件的开发效率

iv. 提高软件的可维护性

v. 提供程序的重用性

二、PHP 自定义函数的语法格式

a) 函数的声明

```
function  函数名()
{
    函数体
}

function  函数名(参数 1，参数 2，参数.... )   //参数列表，如果有多个就作用,分开
{
    函数体
}
```

function 函数名()

{

函数体

返回值；

}

function 函数体(参数列表)

{

函数体

返回值

}

b) 函数名

i. 函数必须调用才能执行，可以在声明之前调用，也可以在函数声明之后调用

ii. 函数名命名和变量一样 aaa bbb ccc aaaBbbCcc, 函数名称一定要有意义（有效的函数名以字母或下划线打头，后面跟字母，数字或下划线）

iii. 函数在声明时不能重名

c) 函数参数

i. 可以通过向函数传递参数，改变函数功能的行为

1. 形参：在声明函数时，声明的参数，参数就是变量，多个参数用逗号分开

2. 实参: 调用函数时传给形参数值（数据，也可以变量），严格按照形参的顺序传实参（对应位置），默认值参数尽量放在最后。

d) 函数返回值

i. 如果没有返回值则称为过程。

ii. 通过使用 return 语句返回数据。

iii. 函数执行到 return 语句就结束，所以不要在这个语句后写代码,也可以使用

return 结束函数的执行。

e) 函数名的作用

i. 调用函数，开始执行函数

ii. 可以向函数中传递数据

iii. 接收返回值

三、PHP 的变量的范围

a) 局部变量： 在函数中声明的变量就是局部变量，只能在自己的函数内部使用。在函数外无法使用 ,因函数在调用完成时就释放内部的变量(结合 C 语言深入理解），要要想在外部使用函数内部的变量（引用[指针]，return，闭包[PHP]）

b) 全局变量： 在函数外声明，在变量声明以后的，直到整个脚本结束前都可以使用, 包括在函数中和 { } 中都可使用，脚本结束才释放

c) 参数就是局部变量，这个局部变量可以调用时去赋值。

d) 在 PHP 中使用全局变量 要通过 global 关键字将这个全局变量包括到函数中才能使用到，在 global 声明之后才是使用全局的变量。

e) 静态变量

i. 静态变量只能声明在函数中（类中），不能在全局声明

ii. 使用 static 在变量前标识

iii. 声明静态变量是在内存初始化静态段分配内存的，脚本结束才释放内存

1. 作用：一个变量可以在同一个函数在多次调用中共享使用该变量。

2. 静态变量在静态代码段中保存并保存该函数的标识

3. 一个函数多次调用之间共用，但只在第一次调用函数时声明到内存，以后再调用函数时，一看该变量是静态变量，就先到静态区中去找，如果有就

不用再声明，而直接使用。在静态区声明的变量不随函数结束而释放内存

```
function test(){
    static $a=0;

    $a++;

    echo $a."<br>";
  }

 test();
 test();
 test();
 test();
 test();
```

输出结果：

1
2
3
4
5

解释：第一次调用函数时声明变量为静态变量，函数调用结束后不释放直到脚本程序后才释放。

```
function test(){
    static $a=0;

    $a++;

    echo $a."<br>";
  }


 function demo(){
    static $a=0;

    $a++;

    echo $a."<br>";
  }

test();
```

```
test();
demo();
demo();
demo();
demo();
test();
test();
test();
test();
```

输入结果：（分配内存静态变量时还保持该静态变量对应函数的标识）

```
1
2
1
2
3
4
3
4
5
6
```

四、PHP 的函数

a) 了解一个函数

i. 函数的功能描述 ---- 决定是否使用这个函数

ii. 函数参数 -----决定函数怎么调用

iii. 函数的返回值 -----调用后怎么处理这个函数

b) 参数类型

i. 常规函数

bool copy（string source, string dest）

ii. 带有 mixed，mixed 表示可以传任何类型的数据

bool chown（string filename, mixed user）

iii. 带有&参数的函数， 表示引用赋值， 这个参数不能传值，只能传一个变量，然后函数将变量的值改变，我们在使用这个变量时，值也是变化的。（涉及到

指针，在 PHP 中是引用，只有在内存中的变量，才有地址，通过&符号可以取出变量的地址。）

bool arsort (array &array [, int sort_flags])

iv. 默认函数 带有[]的函数，表示这个参数是可选的，如果你传值了就使用你传的值，如果没有传值则使用默认值

1. 直接在声明函数时，就为参数给初值。
2. 可选值和必须值， 必须从后向前设置

bool arsort (array &array [, int sort_flags])

v. 带有...的参数函数， ...表示可以传任意多个参数

int array_unshift (array &array, mixed var [, mixed ...])

vi. 回调函数 带有 callback， 就是调用这个函数时需要我们传一个函数进来（函数名，函数名字串）

array array_filter (array input [, callback callback])

c) PHP 默认参数函数

i. 默认参数一定要放在后面

d) PHP 可变个数参数的函数

i. 函数在定义时形参的个数比调用时的实参少，在函数调用时可以使用以下函数在定义函数内部获取实参的参数值。

ii. PHP 函数可变参数列表的实现方法主要是利用 func_get_args()、func_num_args()、func_get_arg()这三个系统函数来实现的

iii. 简化用户传参数

e) 变量函数

i. 如果将一个函数名称（字符串）赋给一个变量（字符串），然后在这个变量后面加上括号，就会调用这个变量值对应的函数

```
<?php
    function add($a,$b){
        return $a + $b;
    }

    function mul($a,$b){
        return $a * $b;
    }

    $reslut = "add";
    $reslut = "mul";

    $reslut(10,20);     //赋值不同调用不同的函数实现不同的功能

    //目的：将不改变函数参数时只要改变其变量值就可以实现不同功能。
```

五、回调函数（PHP，JS）

a) 在使用一个函数的时候，如果传一个变量，不能解决复杂的问题，就需要将一个过程进入到函数中，改变函数的执行行为。

b) 在函数的调用时，在参数中传的不是一个变量或一个值，而是一个函数，这个函数就是回调函数。

实例：首先查看官方手册 http://php.net/manual/zh/function.usort.php

bool**usort**（array&$array，callable$cmp_function）
array 输入的数组
cmp_function 在第一个参数小于，等于或大于第二个参数时，该比较函数必须相应地返回一个小于，等于或大于 0 的整数。

int *callback*（ mixed$a ， mixed$b ）

```
<?php
function cmp($a, $b)
{
    if ($a == $b) {
        return 0;
```

```
    }
    return ($a < $b) ? -1 : 1;
}

$a = array(3, 2, 5, 6, 1);

usort($a, "cmp"); //参数是函数

foreach ($a as $key => $value) {
    echo "$key: $value\n";
}
?>
```

c) 设计回调函数（涉及变量函数，回调函数知识）

过滤条件：过滤掉偶数，但是函数不太灵活

```
function demo($num){
    for($i=0;$i<$num;$i++){
        if($i%2 == 0){
            continue;
        }
        echo $i.'<br/>';
    }
}
demo(100);


function demo($num，$func){
    for($i=0;$i<$num;$i++){
        if($func($i)){          //使用变量函数
            continue;
        }
        echo $i.'<br/>';
    }
}
function test($a){
    If($a == strrev($a))    //只要改变条件即可
        return true;
    else
        return false;
}

 demo(100，"test");
```

d) 不使用变量函数而是借助系统函数，设计回调函数

```
function demo($num，$func){
        for($i=0;$i<$num;$i++){
            // if($func($i)){            //使用变量函数，$func 必须是字符串
              if(call_user_func_array($func),array($i))){ //使用系统函数，详细查看手册
                    continue;
              }
              echo $i.'<br/>';
        }
  }
function test($a){
     If($a == strrev($a))     //只要改变条件即可
          return true;
     else
          return false;
}
 demo(100，"test");
```

e) 设计回调函数，访问对象中的方法，使用变量函数比较复杂

```
function demo($num，$func){
        for($i=0;$i<$num;$i++){
            // if($func($i)){
           //  当回调全局函数时使用变量函数可以，$func 必须是字符串
              if(call_user_func_array($func),array($i))){ //使用系统函数，详细查看手册
                    continue;
              }
              echo $i.'<br/>';
        }
  }
function test($a){
     If($a == strrev($a))     //只要改变条件即可
          return true;
     else
          return false;
}
```

```
class Filter{
        function one($i){
            if($i%3==0){
                return true;
            }else{
                return false;
            }
        }

        static function two($i){
            if(preg_match('/3/',$i)){
                return true;
            }else{
                return false;
            }
        }
    }
/**
    $filter = new Filter();
    $filter->one();
    Filter::two();
 */

 demo(100，"test");  //全局函数使用变量函数是可以的

 demo(100,array(new Filter(),'one'));    //回调对象中的方法，使用变量函数不行
 demo(100,array('Filter','two'));    //回调对象中的静态方法
```

六、系统函数

a) PHP 可以在函数内部再声明函数

i. 目的就是在函数的内部调用

ii. 就是用来帮助外部函数完成一些子功能的。

b) 遍历目录

列出当前目录的所有文件并去掉 "." ".."

```php
<?php
$dir = "/etc/php5/";

// Open a known directory, and proceed to read its contents
if (is_dir($dir)) {
```

```
    if ($dh = opendir($dir)) {
        while (($file = readdir($dh)) !== false) {
            echo "filename: $file:filetype: ".filetype($dir . $file) ."\n";
        }
        closedir($dh);
    }
}
?>
```

七、递归函数

a) 就是在自己内部调用自己的函数名

```
function demo($num){
    echo $num."<br>";
        if($num>0)
            demo($num-1);//递归函数一定要有条件退出
        else
            echo "--------------------<br>";
        echo $num."<br>";
    }
```

demo(3);

输出结果：

3
2
1
0

0
1
2
3

b) 列出目录下的所有文件并去掉"." ".."

```
<?php
function scan_Dir($dir) {
    $arrfiles = array();
    if (is_dir($dir)) {
        if ($handle = opendir($dir)) {
            chdir($dir);
            while (false !== ($file = readdir($handle))) {
                if ($file != "." && $file != "..") {
                    if (is_dir($file)) {
```

```
                        $arr = scan_Dir($file);
                        foreach ($arr as $value) {
                            $arrfiles[] = $dir."/".$value;
                        }
                    } else {
                        $arrfiles[] = $dir."/".$file;
                    }
                }
            }
            chdir("../");
        }
        closedir($handle);
    }
    return $arrfiles;
}

?>
```

八、PHP 加载自定义的函数库

* require: 用于静态包含 //配置文件，出错会导致致命错误
* include: 用于动态包含 // 根据条件包含文件，出错会警告，不会影响程序执行
* require_once
* include_once

以上函数是指令，可以当成函数使用（带括号）也是命令执行 （直接使用）

九、PHP5.3 以后的新特性匿名函数

a) 匿名函数（Anonymous functions），也叫闭包函数（closures），允许 临时创建一个没有指定名称的函数。最经常用作回调函数（callback）参数的值。当然，也有其它应用的情况。

b) 区别常规函数，变量函数，匿名函数

```
<?php
//常规函数
function func($a,$b,$c){
    return $a+$b+$c;
}

echo func(1,2,3).'<br/>';

//变量函数
```

```
function myfunc($a,$b,$c){
    return $a+$b+$c;
}

$name = "myfunc";
echo $name(1,2,3).'<br/>';

//匿名函数(直接把函数体赋给一个变量)
$fun = function ($a,$b,$c){
    return $a+$b+$c;
}; //一定要加分号结束

echo $fun(1,2,3);
var_dump($fun);//object(Closure)
```

十、PHP 闭包函数（closures）的概念（通过匿名函数实现）

a) PHP 闭包实现主要就是靠匿名函数

b) 将匿名函数在普通函数中当作参数传入，也可以被返回，这就实现一个简单的闭包

c) 通俗说：子函数可以使用父函数中的局部变量，这种行为就叫闭包

i. 内部函数

```
<?php
$a = 6666;
function mymain(){
    //子函数,为父函数提供子功能
    $a = 1000;
    echo '1111111';
    function one(){
        global $a;
        echo '2222222'.$a; //不能使用父函数的变量
    }

    function two(){
        echo '3333333';
    }
    //子功能调用
    one();
```

```
        two();
    }
    mymain();   //输出结果 1111111222222266663333333

    //one(); //调用之前要调用父函数,不太合理
```

d) 闭包的两个特点

i. 作为一个函数变量的一个引用，当函数返回时，其处于激活状态

ii. 一个闭包就是当一个函数返回时，一个没有释放资源的栈区

其实以上两点可以合成一点，就是闭包函数返回时，该函数内部变量处于激活状态，函数所在栈区依然保留。

（1）定义：闭包函数推导

```
function mymain(){

    //子函数,为父函数提供子功能

    $a = 1111;
    //匿名函数
    $one = function ($str){
        echo $str.'2222222'.$a.'<br>'; //不能使用$a 变量，还不是闭包函数
    }; //一定要加上分号
    $one('asdf');//在子函数使用匿名函数
}
mymain(); //结果 asdf2222222

//$one('123456'); //不能在外面调用,闭包要可以在外面调用
```

（2）定义：子函数可以使用父函数中的局部变量，以参数形式

```
function mymain(){

    //子函数,为父函数提供子功能

    $a = 1111;
    $b = 5555;
    //匿名函数
    $one = function ($str) use ($a,$b){    //连接闭包和外界变量的关键字
        echo $str.'2222222'.$a.$b.'<br>'; //使用父函数中的局部变量
    }; //一定要加上分号
    $one('asdf');//子函数可以使用父函数中的局部变量，只是简单闭包函数，不够全
}
mymain(); //结果 asdf222222211115555
```

```
//$one('123456'); //不能在外面调用,闭包要可以在外面调用
```

（3）定义：子函数可以使用父函数中的局部变量

```
function mymain(){
    //子函数,为父函数提供子功能
    $a = 1111;
    $b = 5555;
    //匿名函数
    //$one = function ($str) use ($a,$b){    //连接闭包和外界变量的关键字
    $one = function ($str) use (&$a,&$b){

        echo $str.'2222222'.$a.$b.'<br>'; //使用父函数中的局部变量
        ++$b;
        echo $b;
    }; //一定要加上分号
    $one('asdf');//子函数可以使用父函数中的局部变量

    echo    $b ;    //没有改变$b;

}
mymain();
//结果 1
asdf222222211115555
5556
---5555---
//结果 2 （使用传地址方式）
asdf222222211115555
5556
---5556---
```

（4）定义：一个闭包就是当一个函数返回时

```
function mymain(){
    //子函数,为父函数提供子功能
    $a = 1;
    $b = 5;
    //匿名函数
    $one = function ($str) use (&$a,&$b){    //连接闭包和外界变量的关键字
        echo '<br>'.$str.'kkkk'.$a.$b.'<br>'; //使用父函数中的局部变量
        ++$b;
        echo $b.'<br/>';
    }; //一定要加上分号
    echo    '--'.$b.'--';
    return $one;//一个闭包就是当一个函数返回时
}
```

```
$func = mymain();
$func('aaaa');
$func('bbbbbbb');
$func('ccccccccc');
```

输出结果（如果不使用闭包，当调用$func = mymain();时$b 局部变量会释放掉，导致外部不能使用，使用闭包声明不释放）

```
---5---
aaaakkkk15
6
bbbbbbbkkkk16
7
ccccccccckkkk17
8
```

（5）一个匿名函数可以参数传给函数

```
function mymain($func){
    echo    $func();
}
mymain(function(){
    return 'aaaaaaaaaaaaa';
});
```

总结：

1. 闭包外层是个函数

2. 闭包内部都是函数

3. 闭包会 return 内部函数

4. 闭包返回的函数内部不都有 return（因为这个就真的结束了）

5. 执行闭包后，闭包内部变量会存在，而闭包内部函数的内部变量不会存在

十一、闭包的应用场景（类似 jQuery 方法）

a) 保存函数内部变量安全，以最开始的例子为例，外层函数中变量只有内部函数才能访问，而无法通过其他途径访问到，因此保护了外层函数中变量的安全性。

b) 在内存中维持一个变量，依次如前例，由于闭包，外层函数中的变量一直存在于内

存中，因此每次执行，都会使用到。

十二、PHP 命名空间的概述

a) 生活中比如北京的朝阳区和长春朝阳区 ,那直接叫朝阳区就无法识别是哪个城市的

b) 文件 foo.txt 可以同时在目录/home/greg 和 /home/other 中存在，但在同一个目录中不能存在两个 foo.txt 文件，那么要指定文件名时必须将目录名以及目录分隔符放在文件名之前得到 /home/greg/foo.txt。才能找到对应的文件。命名空间是一种封装事物的方法

c) 不能重新定义：函数，类，常量，命名空间就是解决重新定义的冲突

d) 定义命名空间

解决：类，函数和常量同名冲突的问题。

//定义与系统同名 var_dump 的函数会报错

```
function var_dump($var){
    echo $var;
}
```

//使用命名空间定义与系统同名 var_dump 的函数可以定义

```
<?php
namespace myfunc;
function var_dump($var){
    echo $var.'<br/>';
}
var_dump(11111);//使用自定义的
\var_dump(11111); //使用系统的
```

输出结果：

```
11111
int(11111)
```

//解决函数同名

func.init.php

```
function one(){
    echo '111111111';
}

function two(){
    echo '22222222';
}
```

demo.php //在此文件中使用传统的 include 包含时，同名会报错

```
include "func.init.php";   // 传统方式

function one(){
    echo 'aaaaaaaaaa';
}

function two(){
    echo 'bbbbbbbbbbb';
}
one();

//使用命名空间

namespace myfunc; //前置，相对于指定该文件存放目录在 myfunc 目录下

ini_set("display_errors", "On");
error_reporting(E_ALL | E_STRICT);


include "./func.init.php";//指定命名空间区分了文件

function one(){
    echo 'aaaaaaaaaa';
}

function two(){
    echo 'bbbbbbbbbbb';
}
one(); //aaaaaaaaaa
\one(); //111111111
```

十三、PHP 定义子命名空间

a) 必须在其它所有代码之前声明命名空间，命名空间通过关键字 namespace 来声

明，声明后在后面写的代码就是属于这个空间里的。

```
<?php
namespace MyProject;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }

?>

<html>
<?php
namespace MyProject; // 致命错误 -  命名空间必须是程序脚本的第一条语句
?>
```

b) 定义子命名空间

i. 与目录和文件的关系很象，PHP 命名空间也允许指定层次化的命名空间的名称。因此，命名空间的名字可以使用分层次的方式定义。

```
<?php
namespace MyProject\Sub\Level;

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }

?>
```

上面的例子创建了常量 ***MyProject\Sub\Level\CONNECT_OK***，类 ***MyProject\Sub\Level\Connection*** 和函数 ***MyProject\Sub\Level\Connection***。

c) 同一个文件中定义多个命名空间

i. 定义多个命名空间，简单组合语法 （见手册）

ii. 定义多个命名空间，大括号语法 （主要用于将多个 PHP 脚本合并在同一个文件中，不要在大括号之外编写代码）

iii. 定义多个命名空间和不包含在命名空间中的代码 （将全局的非命名空间中的

代码与命名空间中的代码组合在一起，只能使用大括号形式的语法。全局代码必须用一个不带名称的 namespace 语句加上大括号括起来（匿名的命名空间方式处理））

iv. 定义多个命名空间和不包含在命名空间中的代码 （declare）

```php
<?php
declare(encoding='UTF-8');
namespace MyProject {

const CONNECT_OK = 1;
class Connection { /* ... */ }
function connect() { /* ... */  }
}

namespace { // 全局代码
session_start();
$a = MyProject\connect();
echo MyProject\Connection::start();
}
?>
```

十四、使用命名空间：基础

a) 非限定名称，或不包含前缀的类名称，例如 $a=new foo(); 或 foo::staticmethod();

b) 限定名称,或包含前缀的名称，例如 $a = new subnamespace\foo(); 或 subnamespace\foo::staticmethod();。

c) 完全限定名称，或包含了全局前缀操作符的名称，例如， $a = new \currentnamespace\foo(); 或 \currentnamespace\foo::staticmethod();。

file1.php

```php
<?php
namespace Foo\Bar\subnamespace;

const FOO = 1;
function foo() {}
class foo
{
    static function staticmethod() {}
}
?>
```

file2.php

```php
<?php
namespace Foo\Bar;
include 'file1.php';

const FOO = 2;
function foo() {}
class foo
{
    static function staticmethod() {}
}

/* 非限定名称 */
foo(); // 解析为 Foo\Bar\foo resolves to function Foo\Bar\foo
foo::staticmethod(); // 解析为类 Foo\Bar\foo 的静态方法 staticmethod。
resolves to class Foo\Bar\foo, method staticmethod
echo FOO; // resolves to constant Foo\Bar\FOO

/* 限定名称 */
subnamespace\foo(); // 解析为函数 Foo\Bar\subnamespace\foo
subnamespace\foo::staticmethod(); // 解析为类 Foo\Bar\subnamespace\foo,
                                  // 以及类的方法 staticmethod
echo subnamespace\FOO; // 解析为常量 Foo\Bar\subnamespace\FOO

/* 完全限定名称 */
\Foo\Bar\foo(); // 解析为函数 Foo\Bar\foo
\Foo\Bar\foo::staticmethod(); // 解析为类 Foo\Bar\foo, 以及类的方
法 staticmethod
echo \Foo\Bar\FOO; // 解析为常量 Foo\Bar\FOO
?>
```

注意访问任意全局类、函数或常量，都可以使用完全限定名称，例如 **\strlen()** 或 **\Exception** 或 ***\INI_ALL***。

Example #1 在命名空间内部访问全局类、函数和常量

```php
<?php
namespace Foo;

function strlen() {}
const INI_ALL = 3;
class Exception {}

$a = \strlen('hi'); // 调用全局函数 strlen
$b = \INI_ALL; // 访问全局常量 INI_ALL
$c = new \Exception('error'); // 实例化全局类 Exception
?>
```

十五、1111

十六、1111

十七、11

十八、