昨日回顾

逻辑运算符: && || ! xor . 逻辑运算一定要注意存在短路运算[逻辑与和逻辑或中存在]

条件运算符: ? : 条件运算符可以转换成if语句

位运算符: 熟悉 原码,反码,补码的概念. 按照规则区做可以. |按位或

流程控制: 分支结构(if,switch). 循环结构(while,do-while,for,foreach), 顺序结构(代码从上到下顺序执行)

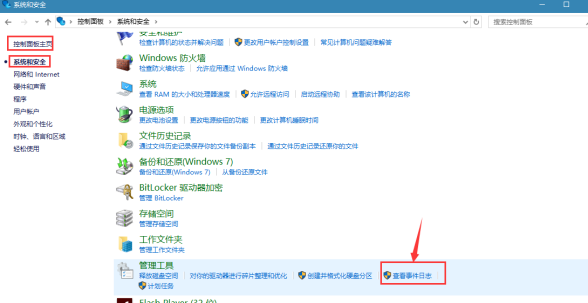
文件载入: include ,require, include\_once ,require\_once

错误处理: 在脚本中,显示和关闭错误 ini\_set(‘display\_errors’,0/1);

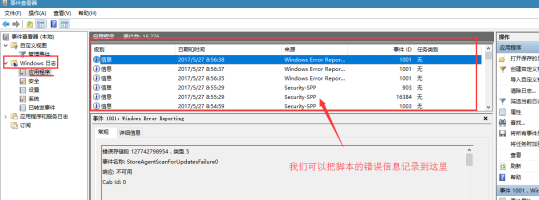
错误写入到日志. 先开启日志功能,再设置日志写入的路径

我们也可以将错误写入到系统的日志

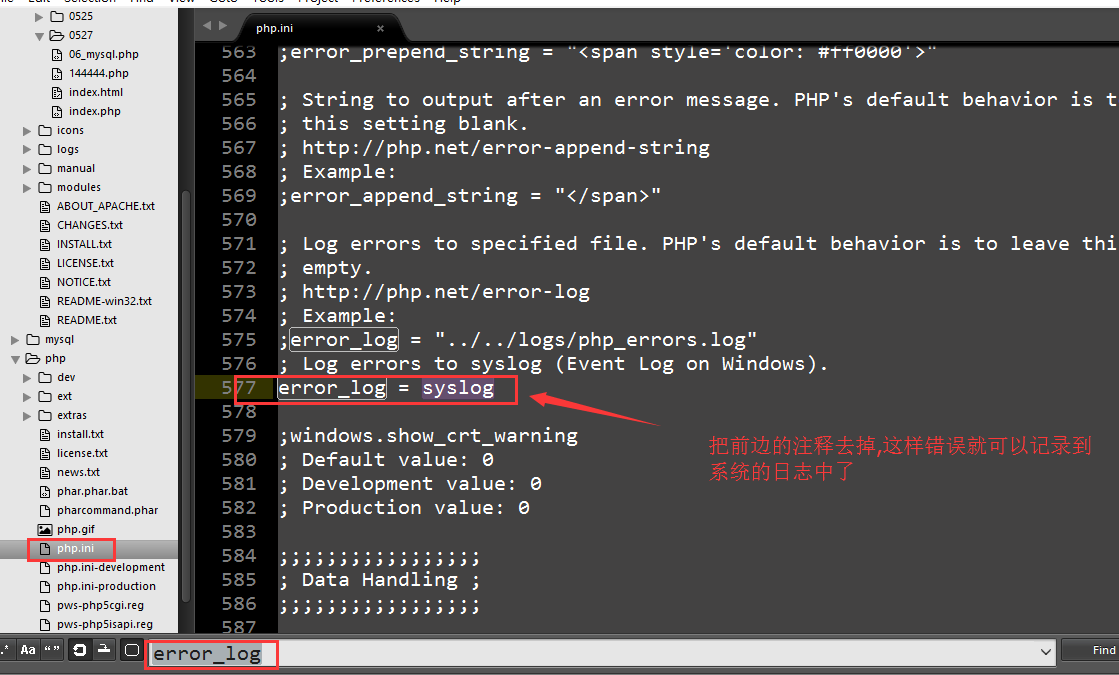
第一步:找到系统的日志



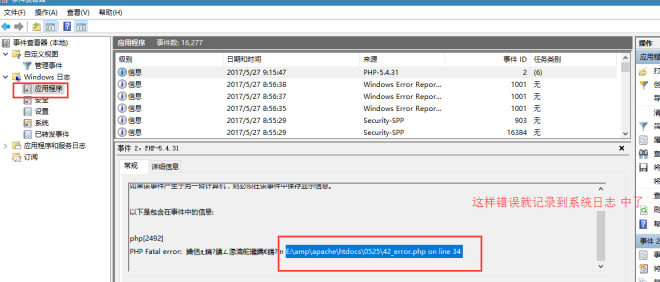
第二步: 查看的地方



第三步: 配置php.ini 文件,搜索log\_errors



第四步: 验证一下

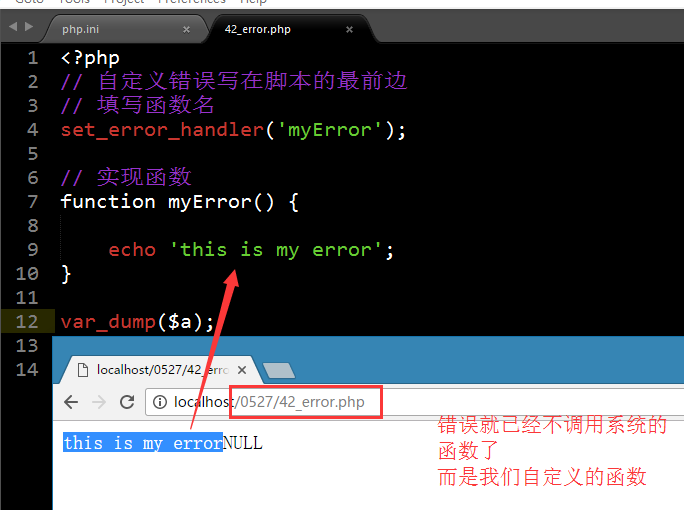


# 第42讲 错误处理

## 自定义错误处理

### 设置用户自定义错误处理函数

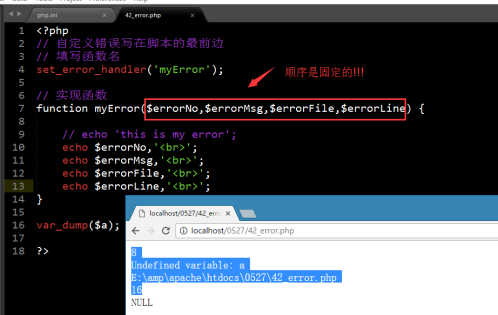
set\_error\_handler ( 错误处理的回调函数名)



### 定义错误处理函数

该函数一般为四个参数，顺序分别为：错误编号，错误提示字符串，出错的文件名，错误行号。4个参数的顺序是固定!!!

错误处理的回调函数名(错误编号，错误提示字符串，出错的文件名，错误行号);



# 第43讲 函数

函数是一个功能独立的代码块,能够实现特定的功能. 我们调用的时候执行

函数分为系统函数和自定义函数

## 函数定义

function 函数名(参数列表) {

函数体

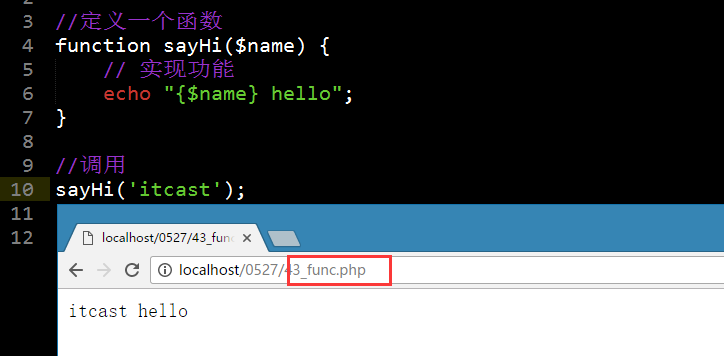
}

## 函数调用

### 调用形式

函数声明完成之后,是需要在需要的时候调用

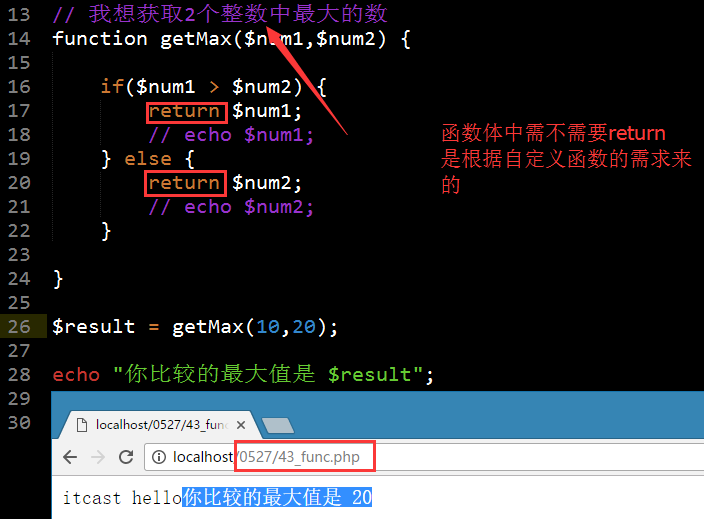
调用是 采用 函数名() 的形式



### 返回值return

如果函数体中有return语句, 这个时候就说明这个函数有返回值.

既然又返回值,我们一般会定义一个变量来 接受这个返回值.

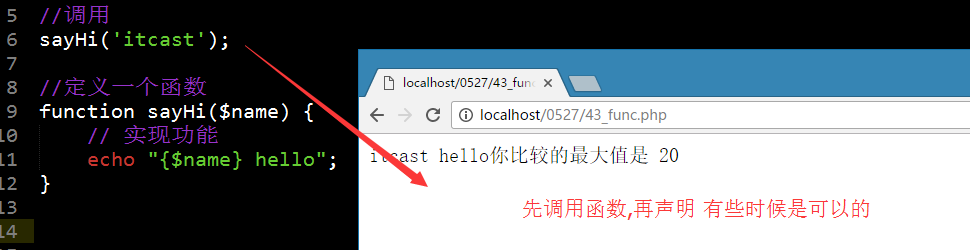


### 注意点

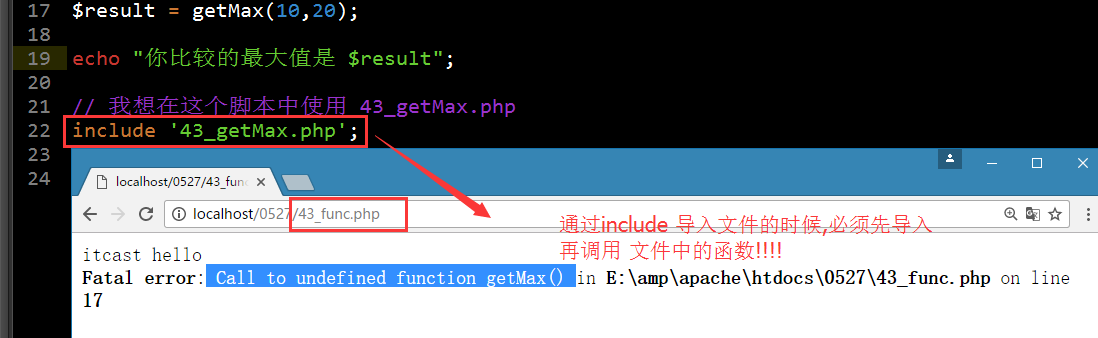
函数是放在代码区.

脚本代码会进行一次预编译,把php的代码预编译成为一个opcode文件.在进行预编译的时候就已经把对应的函数放到了代码区.

函数在调用之前必须先声明. 但是由于我们的php可以进行预编译,所以我们有些时候先调用函数,再声明函数也是可以的.



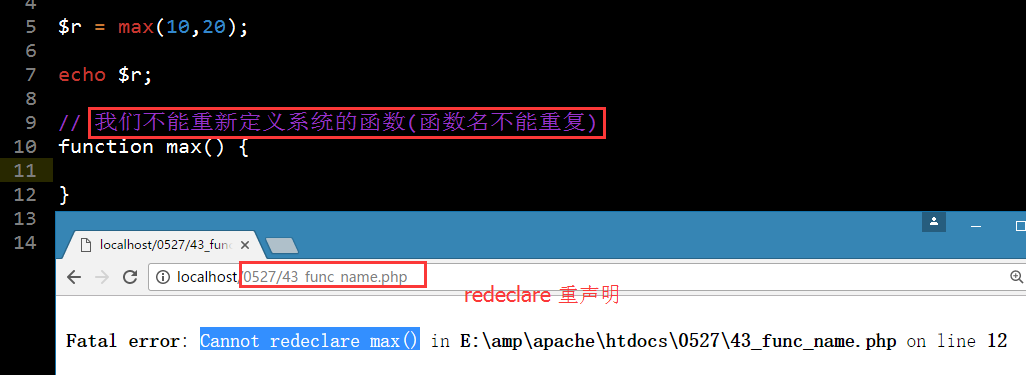
但是, 当我们使用include导入其他文件的时候,必须先声明后调用.



## 函数名

### 命名规范

* 同PHP标识符命名（字母数字下划线组成，不以数字开头）
* 函数不能重新定义。不能使用系统函数。

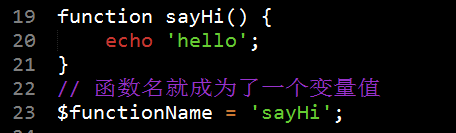


* 函数名不区分大小写, 但是我们的变量名区分大小写
* 函数名一般以 小驼峰 命名 getMax;

### 可变函数

可变变量.

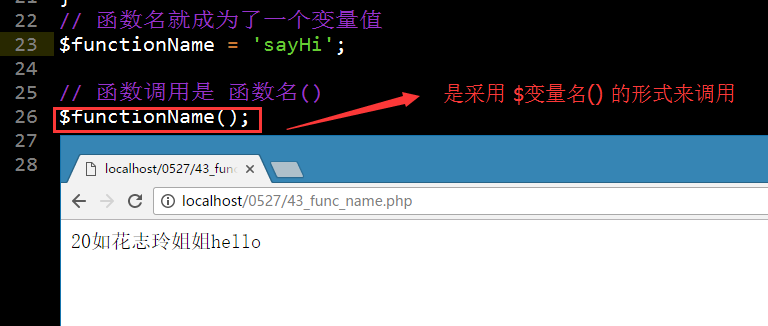
可变函数就是 函数名可以作为一个变量



$functionName 相当于 sayHi;

sayHi函数调用的时候 是采用 sayHi();

所以 $functionName() 相当于 调用 函数,这个函数就是 变量值的函数

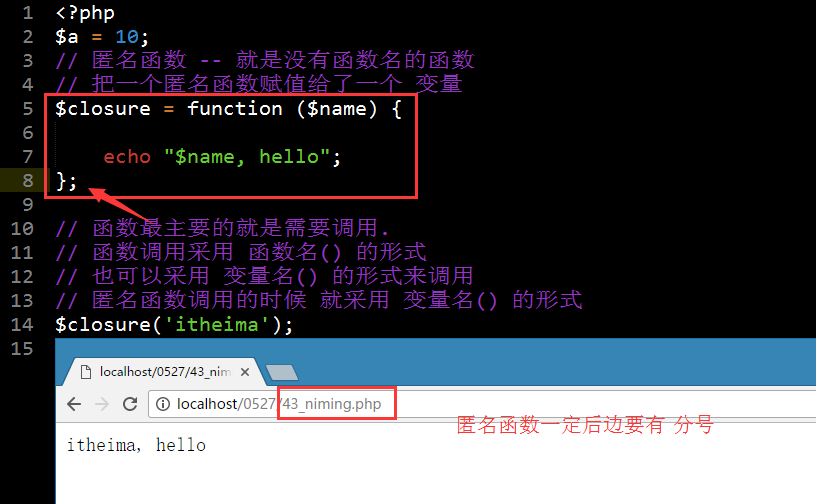


### 匿名函数

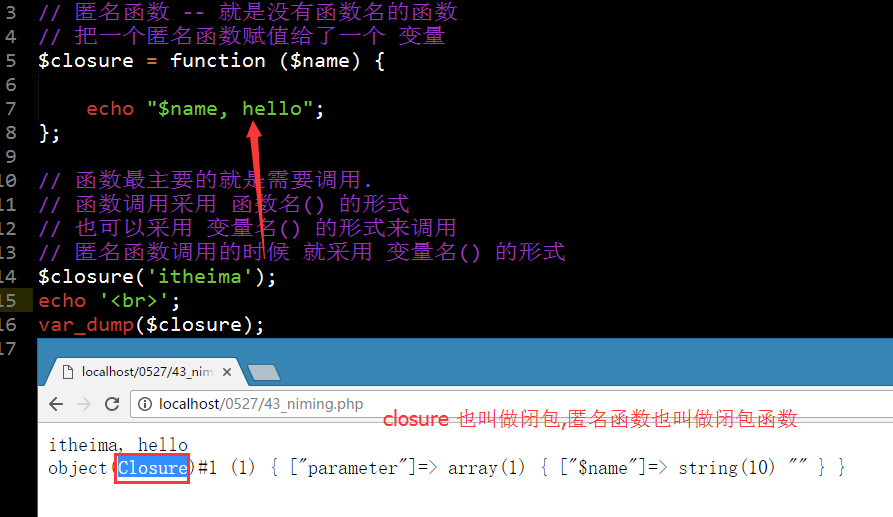
匿名函数就是没有名字的函数,就是没有函数名

因为没有函数名,所以没有办法调用,所以我们将这个匿名函数赋值一个一个变量.因为是赋值操作所以将匿名函数赋值的时候结尾一定要加分号

因为匿名函数赋值给一个变量了,所以匿名函数的调用是采用 变量名() 的形式



匿名函数肯定是先赋值,后调用



# 第44讲 函数的参数

函数其实也可以没有参数,但是如果没有参数,就成为了一个封闭的功能

函数有些时候需要传递有些数据进去,来实现对应的功能.

mysql\_connect 我们需要把ip,端口号,用户名和密码传递进去.

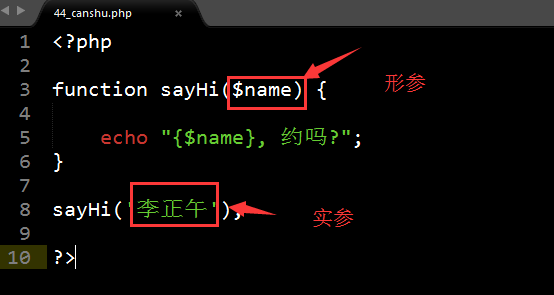
## 形参和实参

### 形参(parmater)

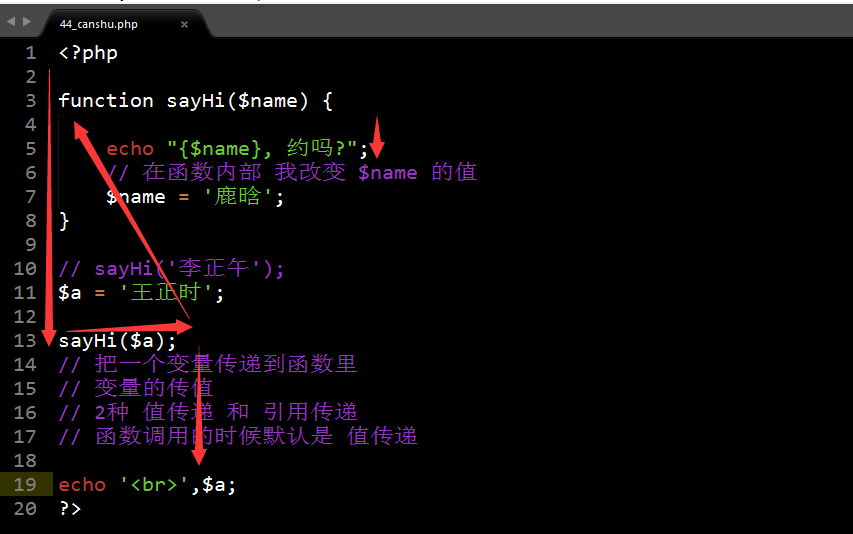
在定义函数的时候规定的，这里的参数只是一个形式，默认的是没有值的，只是说明此处需要使用一个数据来代替！

### 实参(argument)

在调用函数的时候把实际的值传递给该函数，也就是传递给了形参，这个时候，形参才有真正的数据！



## 参数的值传递

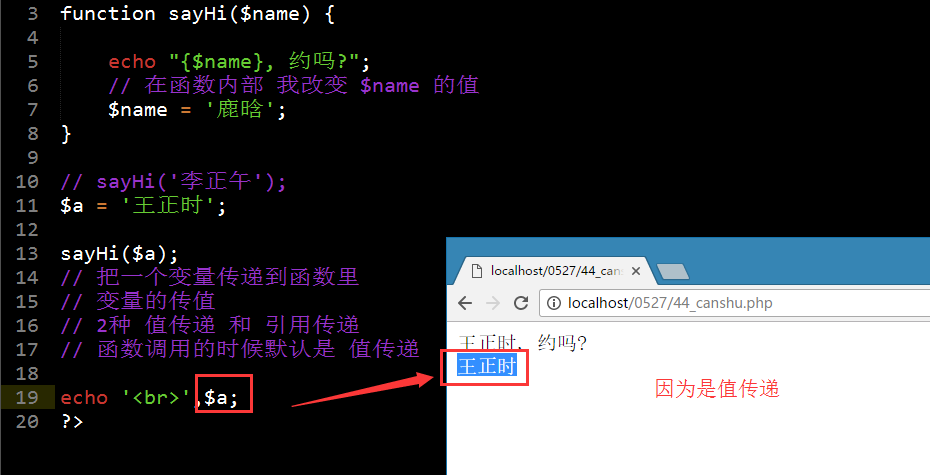


函数在传递值的时候,默认是 值传递.

也就是说 $a = 值 在一个内存区域, $name 又在另外一个内存区域



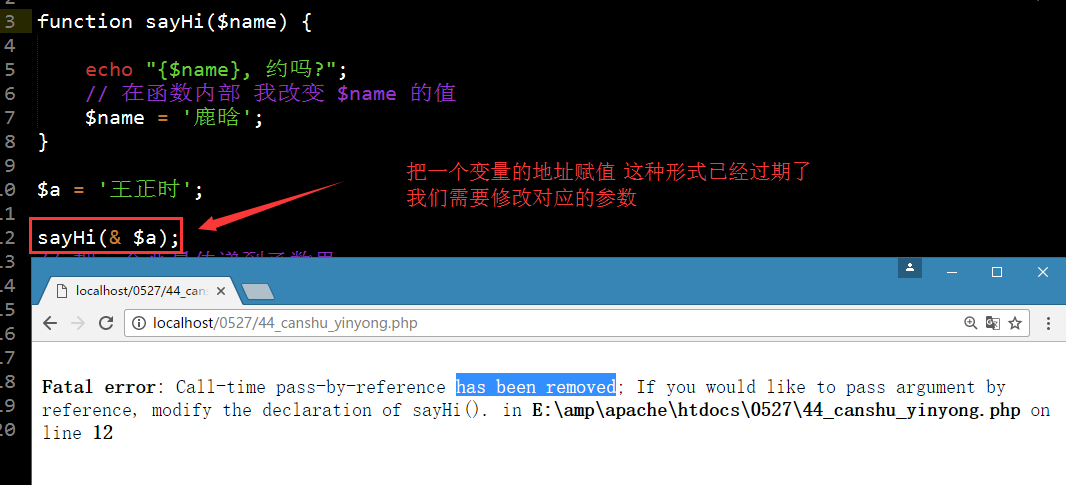
当$name的值改变的时候,不会对 $a 有任何影响



## 参数的引用传递

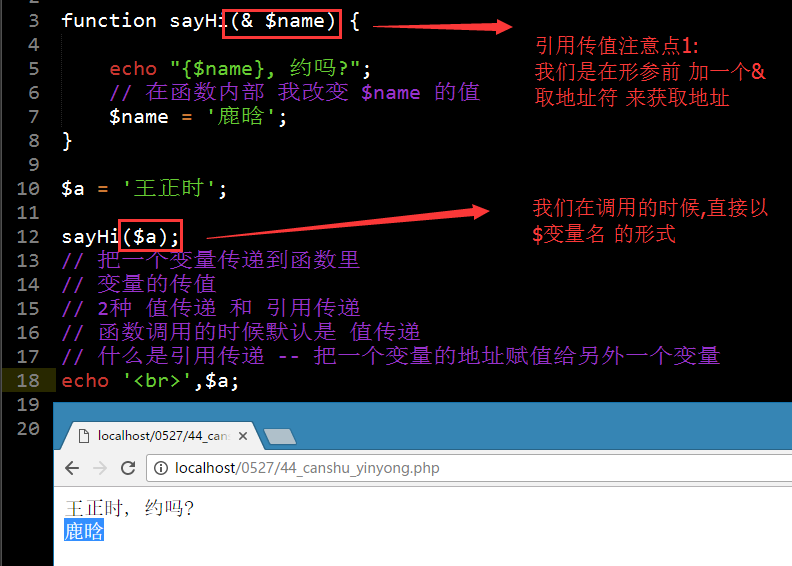
以下截图中的方式已经过去了

sayHi(& 变量名) 这样的形式已经过期了.



现在我们需要修改 对应的形参

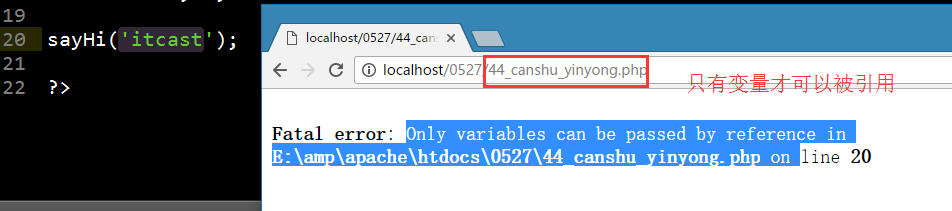
sayHi(& 形参);





注意事项:

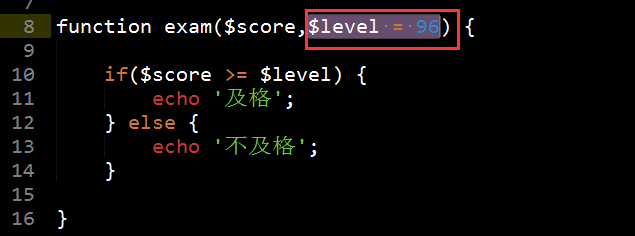
引用传值只能是 变量!!!!



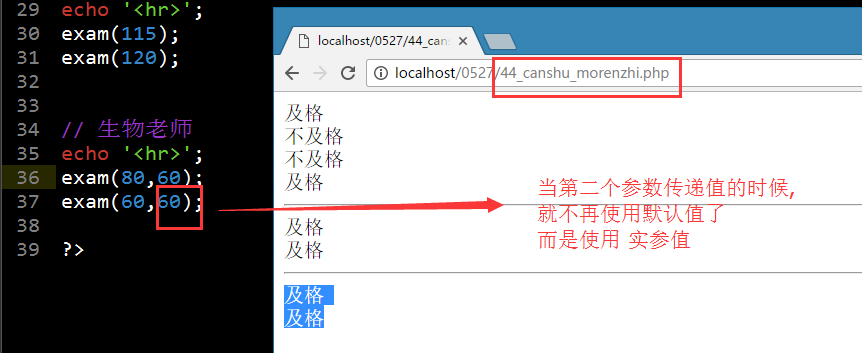
## 参数的默认值

一般来说，形参值应该是由实参传递过来的，但是如果某些参数的值有比较常用的情况，这个时候可以为该形参设置一个默认值！

那么，在调用函数的时候，如果省略掉了该有默认值的形参，该形参的值就使用默认值，如果没有省略，就使用实参传递的值！



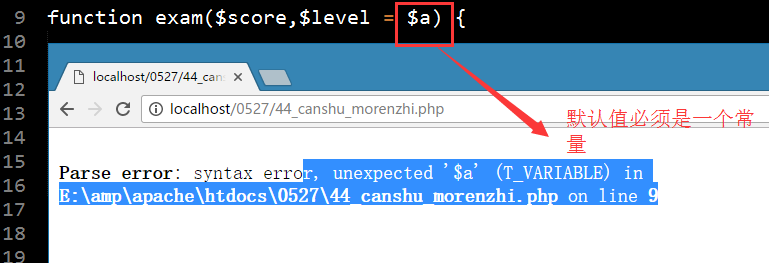
如果没有省略，就使用实参传递的值



注意事项:

一定要把有默认值的参数放到最后.

默认值必须是一个常量,不能是一个变量

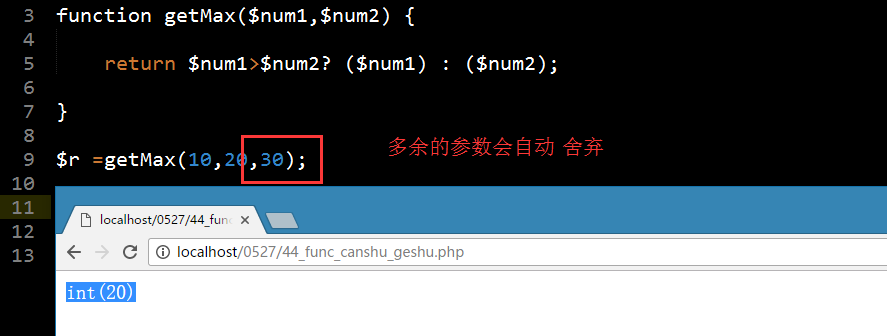


## 参数的数量问题

一般来说，实参和形参的数量应该是相等的，也就是一一对应！

### 实参多于形参

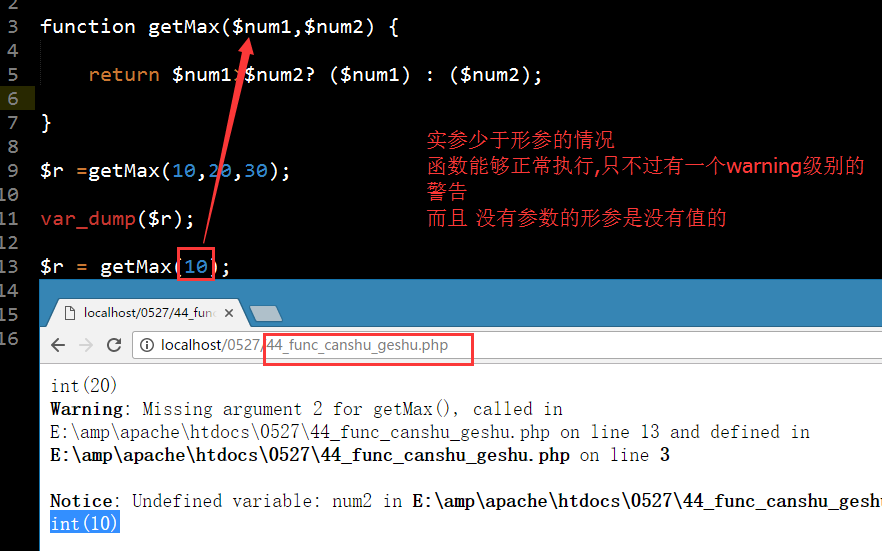
可以正常运行，所以的形参按顺序接收实参的数据，多余的自动舍弃！



### 实参少于形参

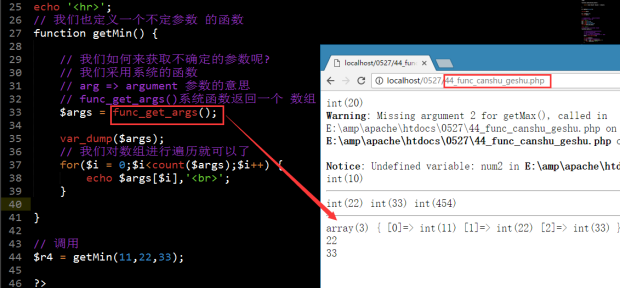
只有一种正常的情况，就是形参列表的最后有默认值的情况！也就是说，有默认值的形参可以不用赋值！！

除此之外，一般会报告一个warning级别的错误，但是脚本会继续运行，只是有部分形参没有获取到数据而已！



### 不定参数的函数

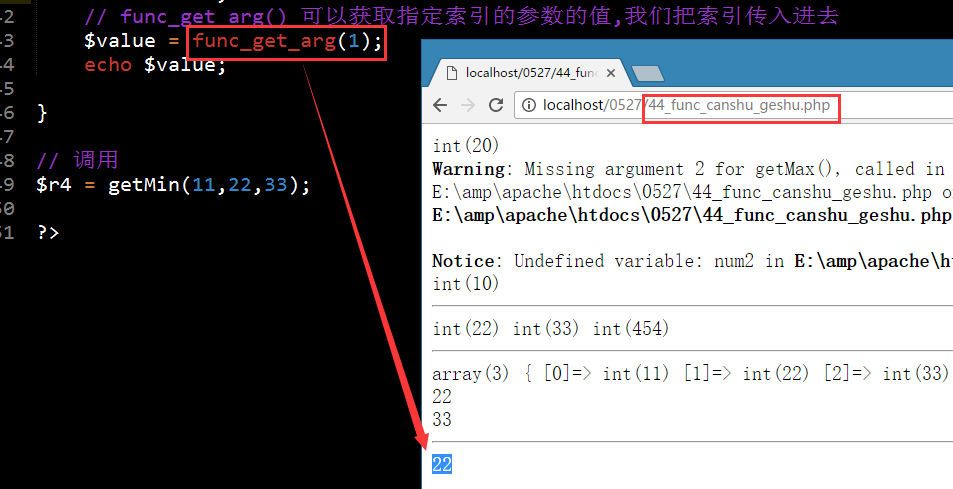
func\_get\_args 能够获取到所有的参数,返回是一个数组



我们可以通过对数组进行遍历来获取所有的参数.

我们也可以通过 $数组名[索引] 的形式获取指定的参数

我们也可以通过系统函数 func\_get\_arg(index) 来获取指定的索引的参数值



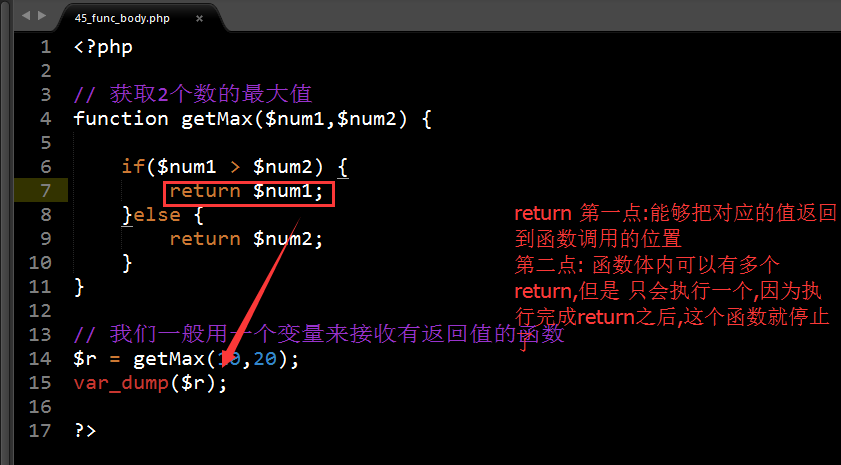
# 第45讲 函数体

## 概念

就是可执行代码的集合，以完成相应的功能！

## return语句

* 将函数的返回值，返回给调用该函数的位置！
* 停止函数的运行！



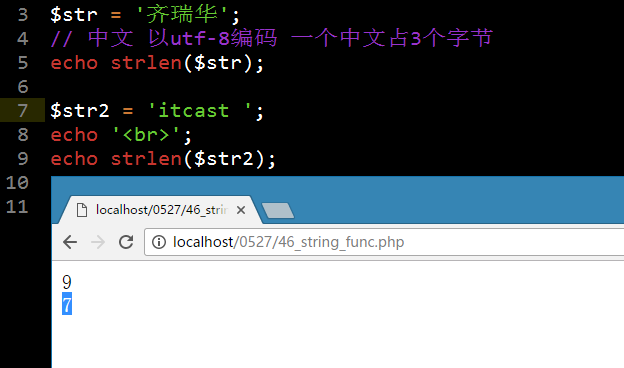
# 第46讲 字符串函数

字符串的4种表现形式: 单引号,双引号, heredoc定界符,nowdoc定界符

## strlen

字符串的长度.



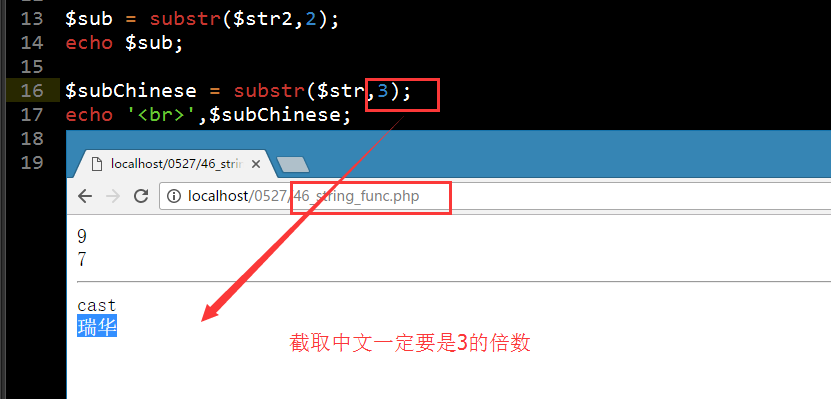


## substr

截取字符串

一个字符串,从哪里开始,截取到哪里.





## strtolower

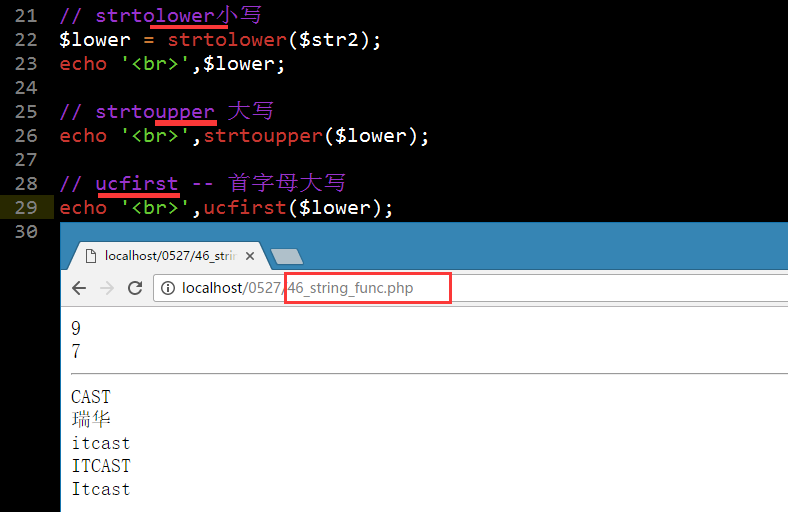
字符串都变成小写

## strtoupper

字符串都变成大写

## ucfirst

首字母大写



## strrev

rev表示 reverse 反转的意思

将字符串反转过来

## strpos

pos 表示 position位置

字符串从左到右开始查找,查找指定字符第一次出现的位置

itcast

## strrpos

字符串反转过来从右往左查找

## strchr

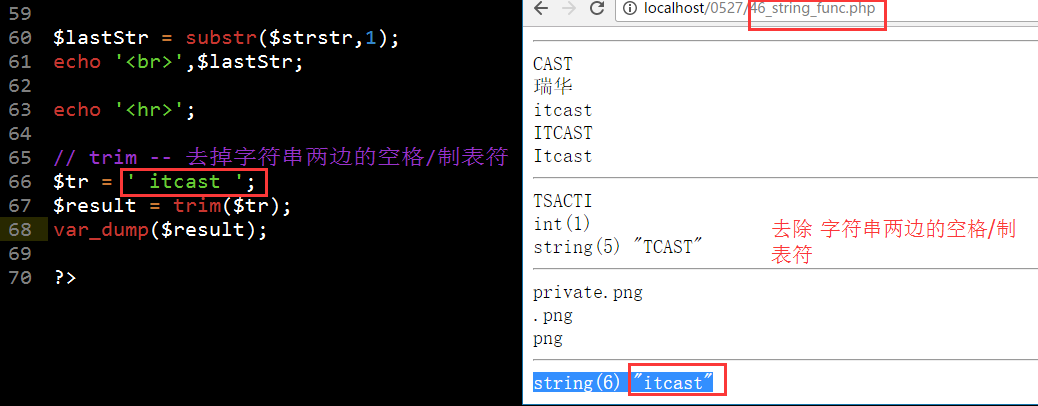
strchr 也是 strstr函数



## strrchr

## trim

用户登录注册的时候,会填写信息. 信息的开始和结束的空格 是需要去掉的

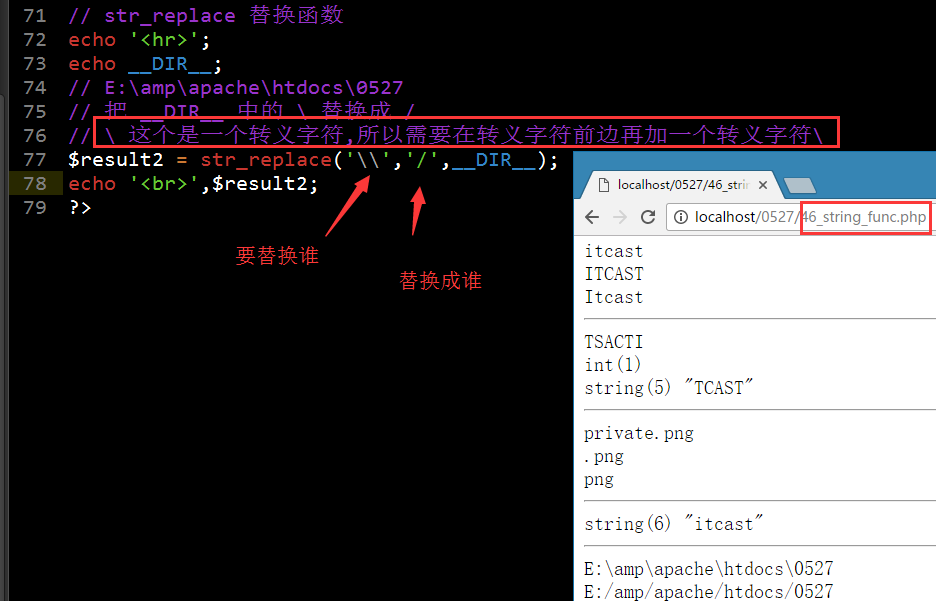


左边 left 去除字段串左边的空格用 ltrim

右边 right 去除右边的空格用 rtrim

## str\_replace

字符串替换函数



# 第47讲 时间函数

## time

从1970年1月1日 到现在的一个秒数

## date

日期函数

date分为两部分,前半部分是用双引号包裹起来的,其中包裹起来的英文字母具有一定的含义.

后半部分,就是秒数

记住6个字母的日期含义:

年 Y

月 m

日 d

时 H

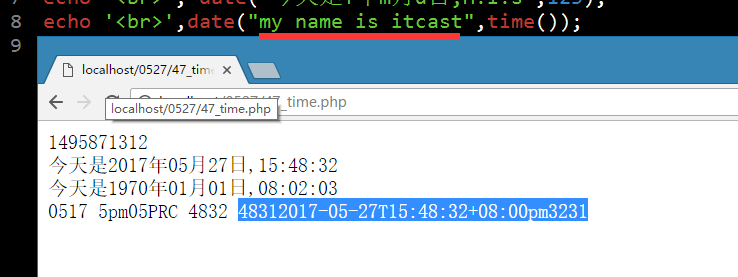
分 i

秒 s

date(“今天是Y年m月”,time());

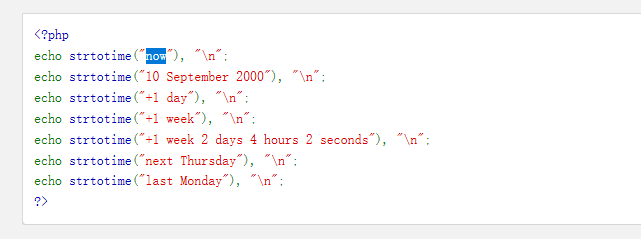
date前半部分里的英文字母都有一定的含义,千万不要随便写





## strtotime

英文中有一些表示时间 的单词, today,yesterday ,next week;



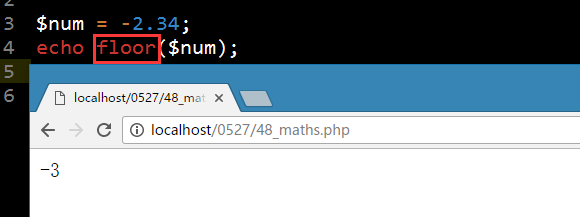
# 第48讲 数学函数

## abs

取绝对值

## floor

向下取整



## ceil

向上取整

## round

四舍五入

## rand和mt\_rand

rand获取随机数,rand的随机数范围包括 写入的这2个值. rand(0,100). 0 和100包括在随机数中.

mt\_rand和rand一样,只不过 mt\_rand的运行效率高一点,推荐大家使用mt\_rand

# 第49讲 变量的作用域

## 概念

指的是当一个变量在定义完之后，在哪里可以使用在哪里不可以使用的问题，也就是一个变量的有效范围！

## 局部作用域和全局作用域

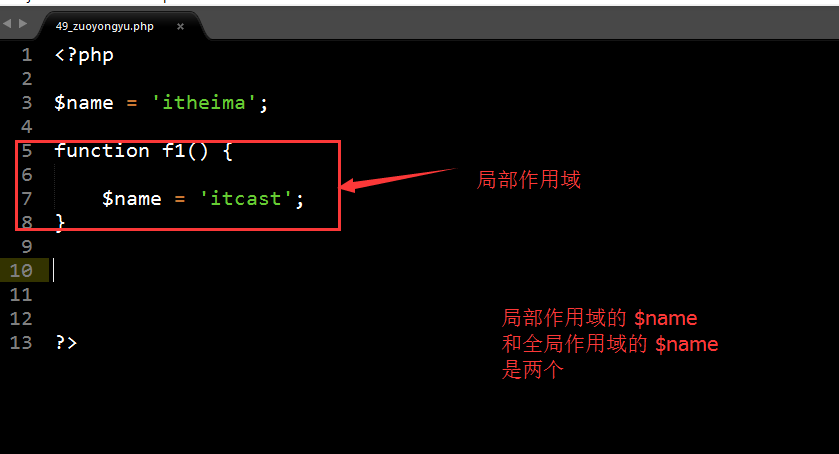
变量的作用域就有2个: 局部作用域 和 全局作用域

### 局部作用域

在函数内部定义的变量就是 局部作用域.这个变量只能在局部作用域内使用(只能在函数的内部使用)

### 全局作用域

在函数外部定义的变量就是 全局作用域.



### 注意事项



① 局部作用域与全局作用域 没有任何交集

② 局部作用域 与局部作用域之间也是没有任何交集

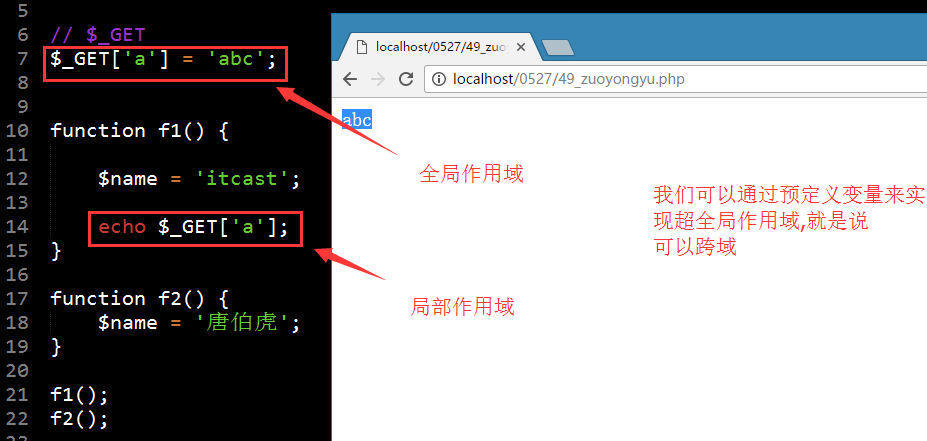
## 超全局变量

超全局变量就有点和js里的全局变量类似.

预定义变量有9个,这9个预定义变量都是 超全局变量.

预定义变量都是数组,

给数组添加元素 $数组名[‘名’] = 值; 的形式来添加

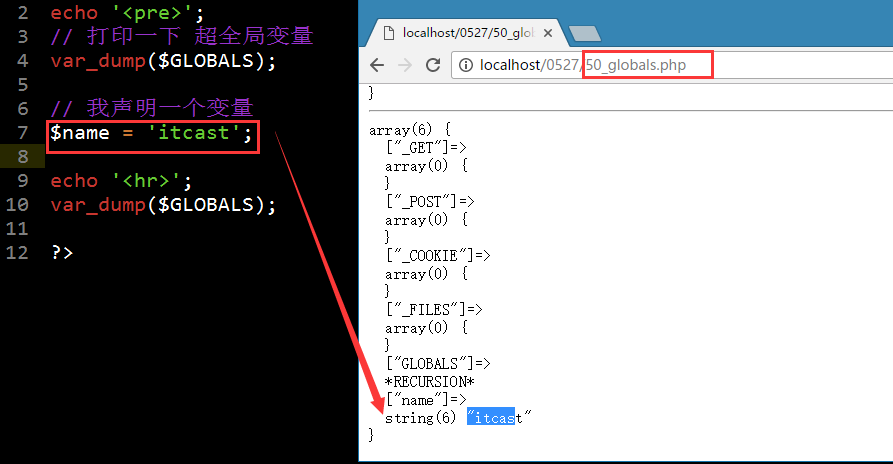


但是 这9个预定义变量都有自己的工作,唯独 $GLOBALS 用于我们作用域

# 第50讲 超全局变量$GLOBALS

## 语法

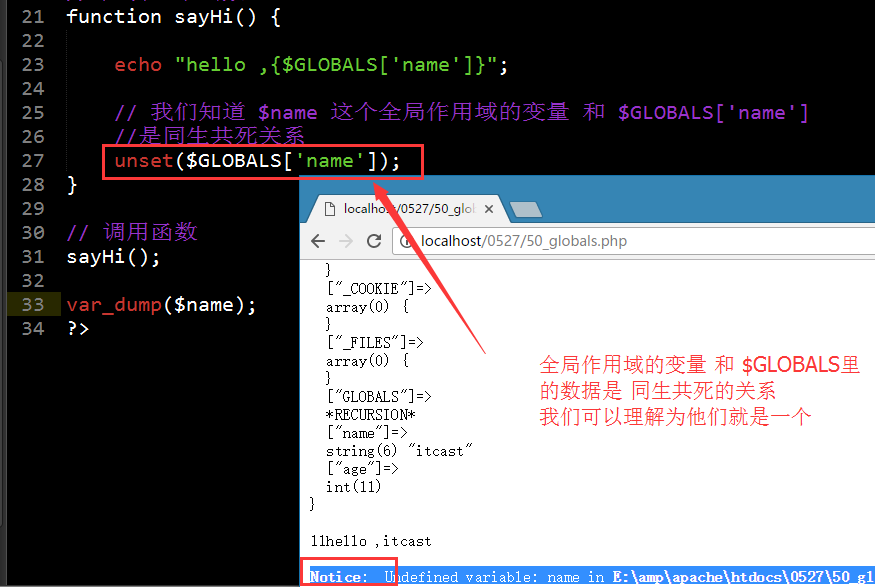
① 当我们在全局作用域中,声明一个变量,声明完成之后,对应的也会在$GLOBALS中多一个元素,这个元素的名 = 变量的名, 这个元素的值 = 变量的值;



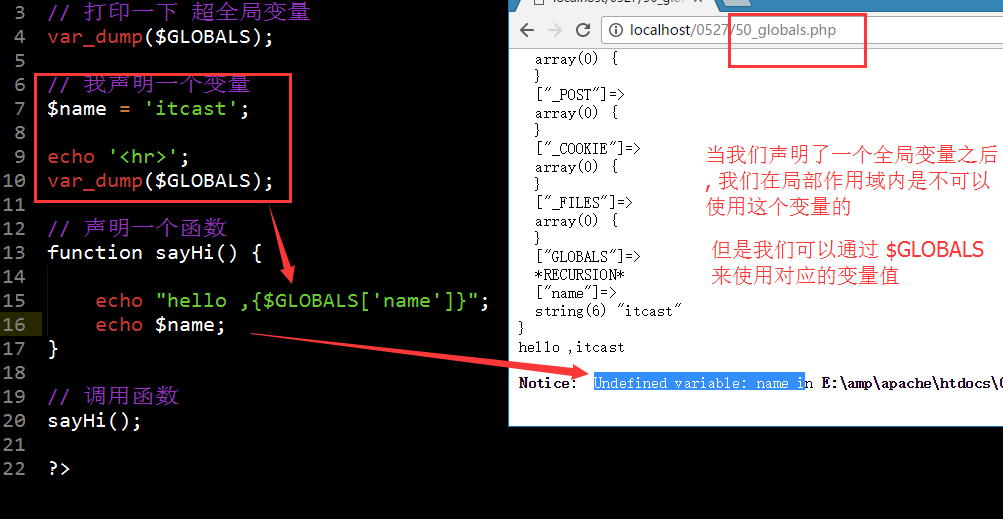
② 我们通过$GLOBALS来添加一个元素,对应的也会在 全局作用域中 多一个变量,变量的名是元素的名,变量的值是元素的值



③ $globals里是数据 和 我们的全局作用域的变量的关系 是同生共死的关系. 我们只要删除一个/改变一个,另外一个也会发生改变



全局变量的使用

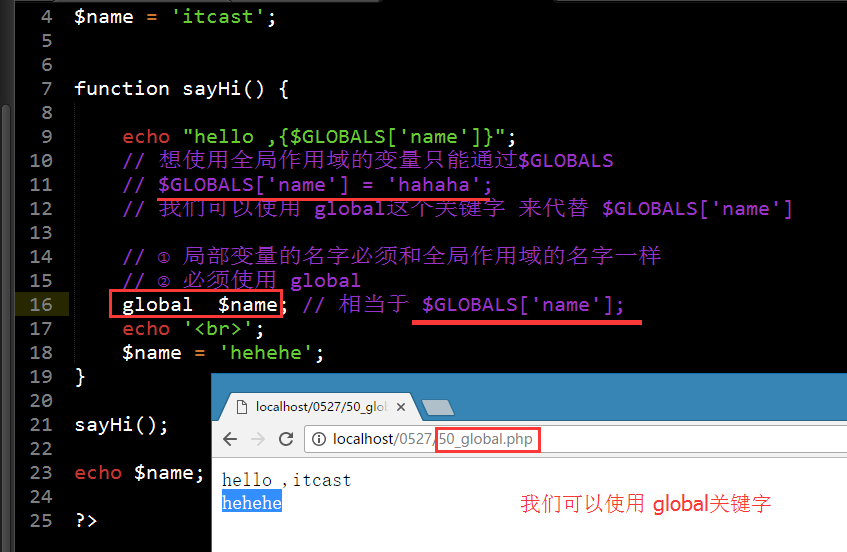


## 关键字global

global $变量名 相当于 $GLOBALS[名]

注意两点: 第一, global后边的变量名必须和 全局作用域的变量名一样

第二,必须使用 global关键字



# 第51讲 变量的生命周期

## 概念

变量什么时候出现（“诞生”），又什么时候消失（“死亡”）

我们声明的时候就诞生了.

unset的时候可以让变量死亡

脚本执行结束之后变量也就死亡了

局部作用域的变量会在函数调用结束之后死亡



## 静态局部变量

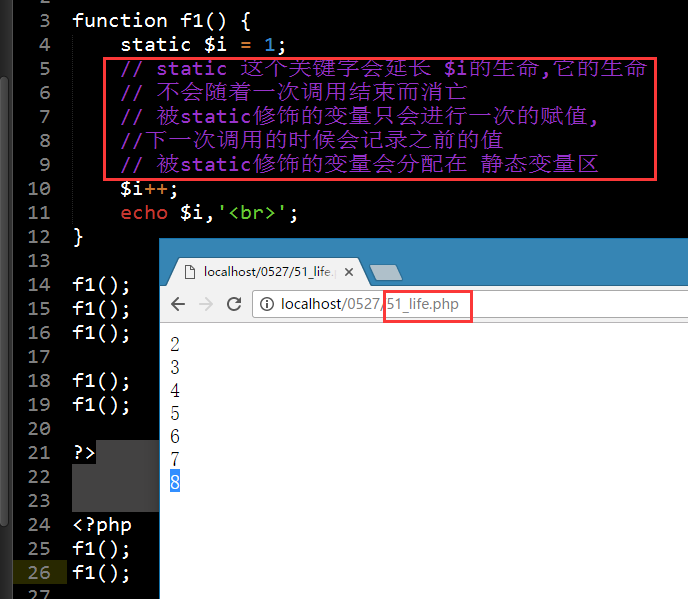
### 概念

不随着函数执行的结束而消失，而是，只要函数存在（只要脚本周期还没有结束），该变量就一直存在（生命更长）！

关键字是 static

static $变量 = 初始值;

被static修饰的变量,第一个会有一个初始值,下一次被调用的时候会记录之前的值,所以不会重复的赋值



### 注意事项

无论有没有这个static关键字，函数内的变量都是局部变量，函数外同样都不能使用！

# 第52讲 递归调用

更多是一种算法. 我们在解决大问题的时候,会把大问题分成成一个一个的小问题,然后我们解决这些小问题,解决小问题发现这些小问题还需要像刚才一样把这个小问题再分解成小问题.

## 递归含义

递归调用实质就是 自己调用自己的过程.

2个例子: 不断在调用自己,什么时候自己开始调用自己

调用自己肯定有一个终结的时候

递归的2要素:

1. 递归是从什么时候开始的

2.递归是从什么时候结束的

## 案例:数列

已知数列的前2项,前两项分别为1和1.

第3项 = 第2项 + 第1项;

第4项 = 第3项 + 第2项;

第5项 = 第4项 + 第3项;

…

第n项 = 第n-1项 + 第n-2项;

1 1 2 3 5 8 13 21 34 55 89 …

递归的开始点就是 不断重复的开始点

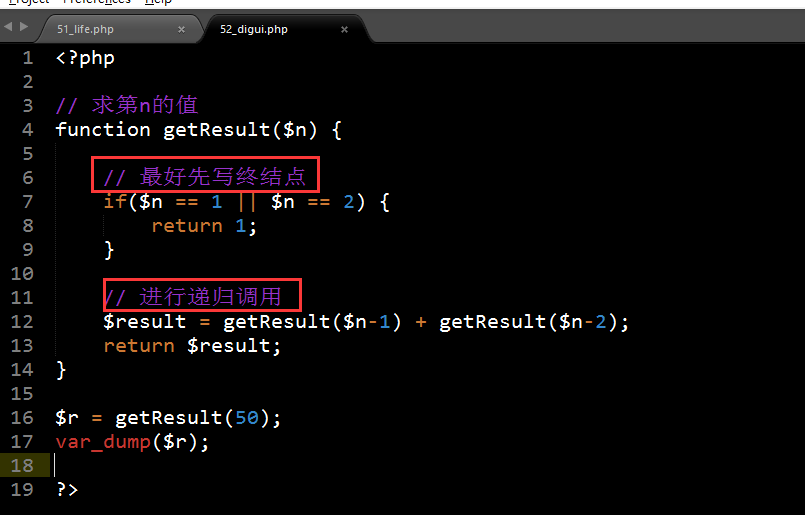
递归的终结点其实就是 已知条件

1 n = 1|n=2

f(n)

f(n-1) +f(n-2) n>2

写代码的时候最好先写终结点



递归的原理是:



作业题:

阶乘.数学知识

1! = 1

2!=2\*1

3!=3\*2\*1

4! = 4\*3\*2\*1

5! = 5\*4\*3\*2\*1;

6! = 6\*5\*4\*3\*2\*1;

…

n! = (n)\*(n-1)\*(n-2)\*…4\*3\*2\*1

先找个规律,找完规律之后用递归!!