昨日回顾

进制

整型: 内存中占用4个字节. 整形在内存中都是以二进制补码的形式存在

原码: 二进制的最高位为符号位:0表示正数 1表示负数

其他位表示数值

反码: 正数的原码,反码,补码都一样,都是它本身.

负数的反码: 符号位不变,其他位按位取反(1变0,0变1)

补码:

正数的原码,反码,补码都是一样,都是它本身.

负数的补码: 反码+1

浮点型:

内存中占用8个字节. 浮点数的有效位数 14位. 浮点数在内存中是以指数的形式存在的.

12345678901234567890 = 1.2345678901234e19

浮点数尽量不要进行比较.

布尔型: 记住6种转换为false的情况就可以: 整数的0,浮点数的0.0,字符串的空’’,字符串的0’0’,空数组array(), null

字符串: 单引号和双引号

定界符(heredoc 没有引号 nowdoc 有引号)

特殊类型:

null

resource

类型转换 : 自动类型转换 和 强制类型转换

字符串转换成数值型 和 其他类型转换成布尔型.

算术运算符: +,-,\*,/,%, 自加,自减

赋值运算符: = 等号是右结合,先计算等号的右边!

字符串运算符: .

关系运算符: <= >=,

==

===

# 第28讲 逻辑运算符

## 逻辑与

&& / and

逻辑与的语法含义是，只有两边表达式的值同时为真的时候，整个表达式才为真，只要有一个为假，整个表达式都为假！

|  |  |  |  |
| --- | --- | --- | --- |
| 表达式1 | && | 表达式2 | 结果 |
| True |  | True | True |
| True |  | False | false |
| False |  | True | False |
| False |  | False | false |

## 逻辑或

|| / or

逻辑或的语法含义是，只有两边表达式的值同时为假的时候，整个表达式才为假，只要有一个为真，整个表达式都为真！

|  |  |  |  |
| --- | --- | --- | --- |
| 表达式1 | || | 表达式2 | 结果 |
| True |  | True | True |
| True |  | False | True |
| False |  | True | True |
| False |  | False | false |

## 逻辑非

!

!真 就是 假

!假 就是 真

语法意义：非真即假，非假即真！

## 异或

xor

同性之间不能谈恋爱

1 xor 1 假

0 xor 0 假

1 xor 0 真

0 xor 1 真

语法意义是：两边表达式的值相同为假，不相同为真！

也就是两个假或两个真的时候，整个表达式为假，一真一假的时候，整个表达式为真！

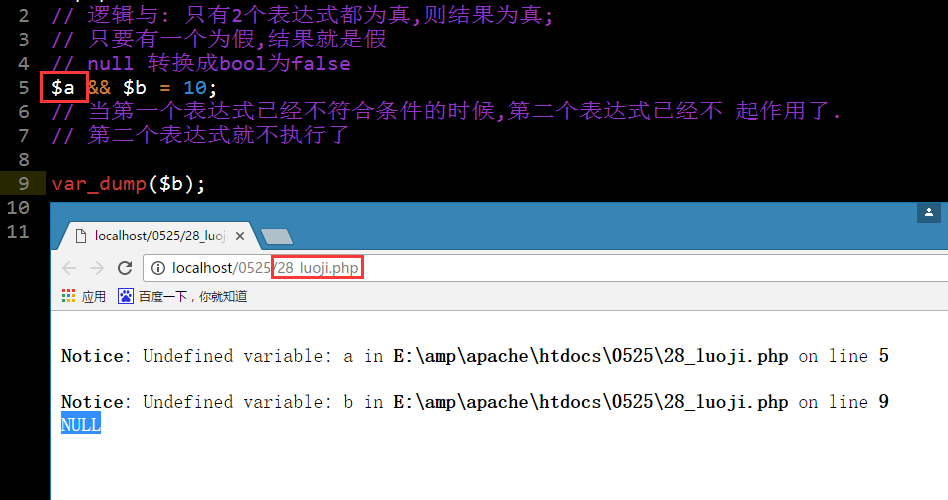
## 短路运算

逻辑与和逻辑或都有短路运算！

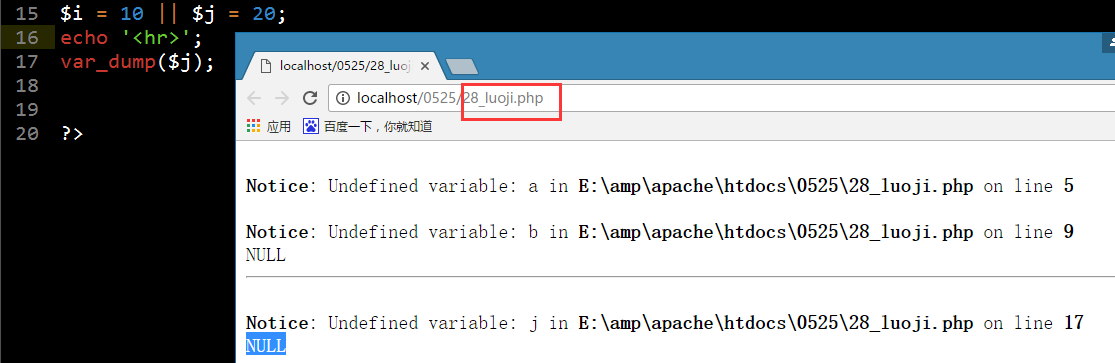
比如逻辑&&： 如果第一个表达式为假，第二个表达式无论是真还是假，整个表达式都为假，所以

如果第一个表达式为假，第二个表达式既不执行也不起任何的作用！

这种现象就是短路运算！



逻辑或中也存在短路运算



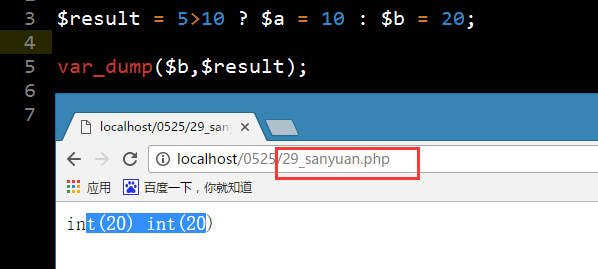
# 第29讲 条件运算符

## 语法形式

表达式1 ? 表达式2: 表达式3;

如果表达式1成立,则 执行表达式2,并且会把表达式2的值返回

如果表达式1不成立,就执行表达式3,并且会把表达式3的值返回



三元运算符一般可以转换成 if语句

if(表达式1) {

表达式2;

} else {

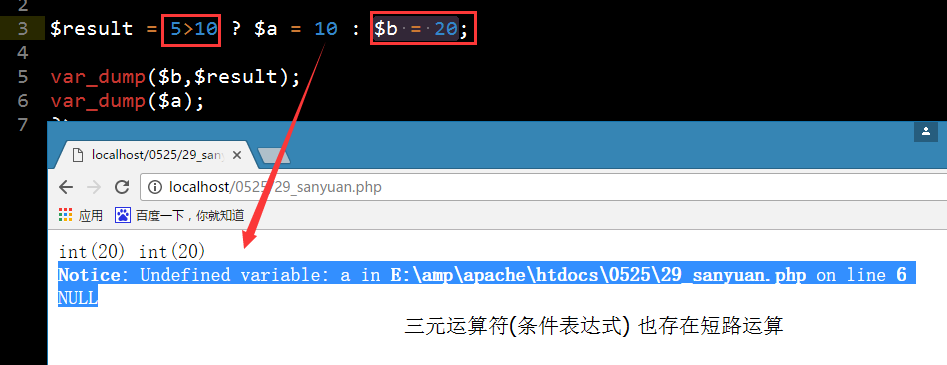
表达式3

}

建议大家使用:条件运算符,因为条件运算符的运行效率高

## 注意事项

条件运算符也存在短路运算!



# 第30讲 位运算符(了解)

## 什么是位运算

这里是位指的就是bit位！

说明了位运算符就是专门针对二进制位上的一种运算符！

一般来说，位运算符主要是针对整型数据！

前面讲过，整型数据是以二进制的补码的形式存放的，在内存中一共占32个bit位！

也就是说，一个整型数据的位运算要计算32次！

而一个bit位只有0和1两种状态，所以位运算有点类似数据中的布尔型中的运算！

## 各种位运算符的含义

& 按位与，表示两个位都同时为1的时候，才返回1，否则就返回0

1&1 = 1

1&0 = 0

0&1 = 0

0&0 = 0

| 按位或，表示两个位同时都为0的时候，才返回0，否则就返回1

1|1 = 1

1|0 = 1

0|1 = 1

0|0 = 0

~ 按位非，非0即1，非1即0

~0 = 1

~1 = 0

^ 按位异或，如果两个位不同时候，返回1，相同时返回0

1^1 = 0

0^0 = 0

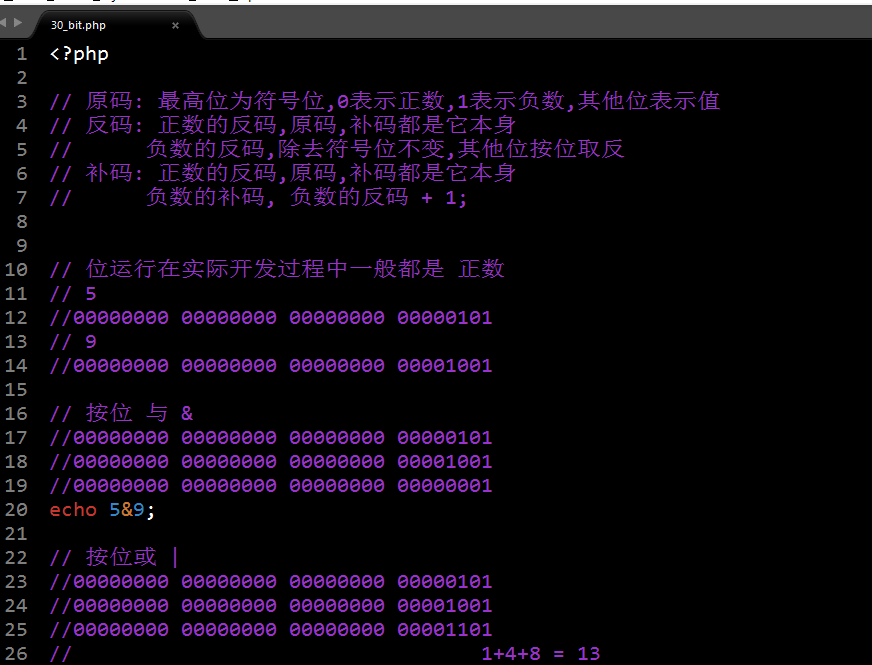
1^ 0 = 1

0^1 = 1

<< 按位左移，将整个字节向左移动，右边用0填充！

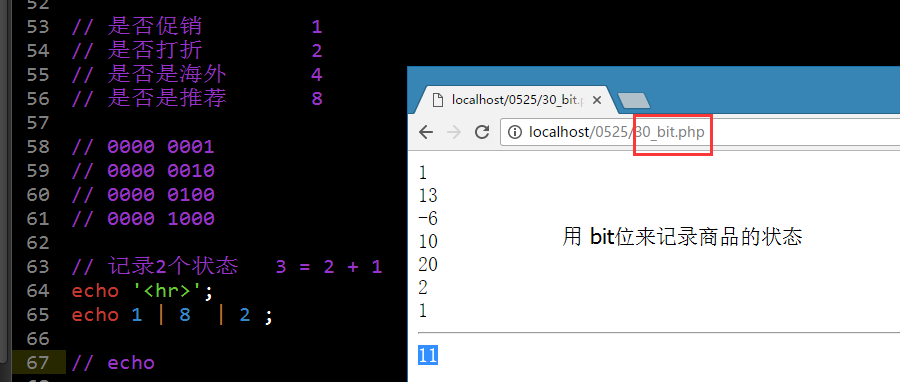
>> 按位右移，将整个字节向右移动，正数左边用0填充，负数的左边用1填充！

## 位运算符举例



## 位运算符在实际开发中的应用

有很多状态我们可以通过bit位来实现

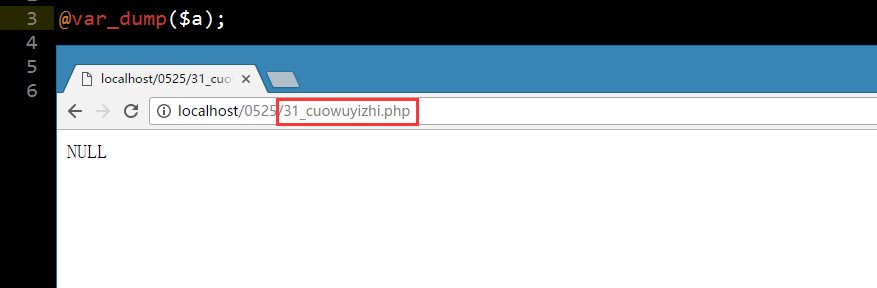


# 第31讲 错误抑制符

## 错误抑制符

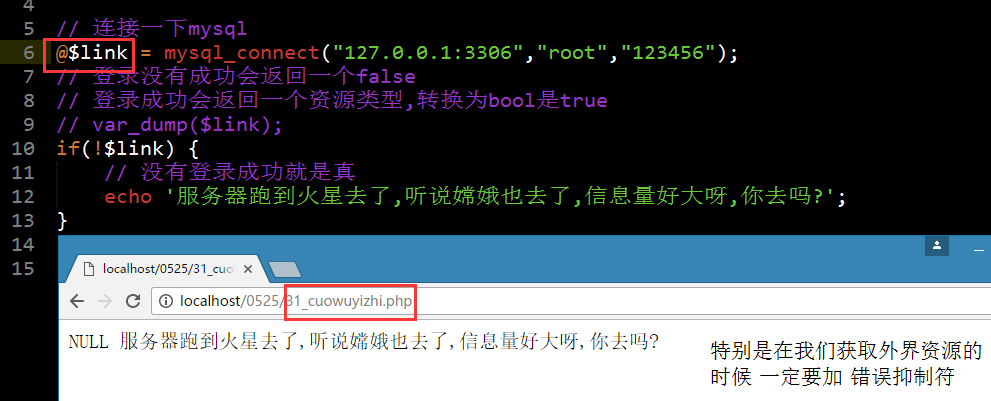
@就是错误抑制符

@可以抑制错误的输出.防止错误在浏览器中显示.



错误抑制符一般我们在获取外界资源的时候使用.

因为我们不知道外界资源会发生什么问题



# 第32讲 流程控制简介

所谓的程序的流程控制，就是指程序执行的“路线”！

一般我们是用流程图来描述程序的流程问题！

## 流程图基本符号(了解)

开始和结束： 椭圆



流程走向：箭头



语句（块）执行： 长方形



条件判断： 菱形



## 顺序结构

顺序结构是整个脚本的最基本的结构！

## 选择结构

也叫分支结构，在程序的运行过程中出现了两个或者两个以上的分支，而只能根据相关的条件选择其中的一个分支执行！

if

switch

## 循环结构

在满足一定的条件的时候，重复执行一段代码！

while循环

do-while循环

for循环

foreach循环（用于数组）

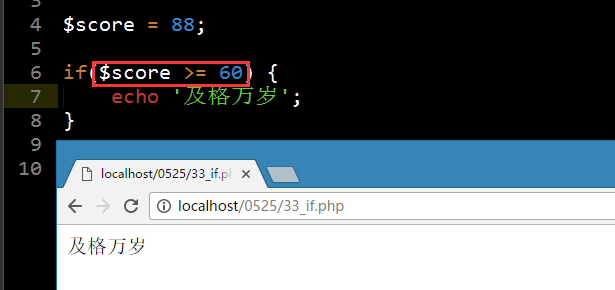
# 第33讲 if语句

## 条件执行

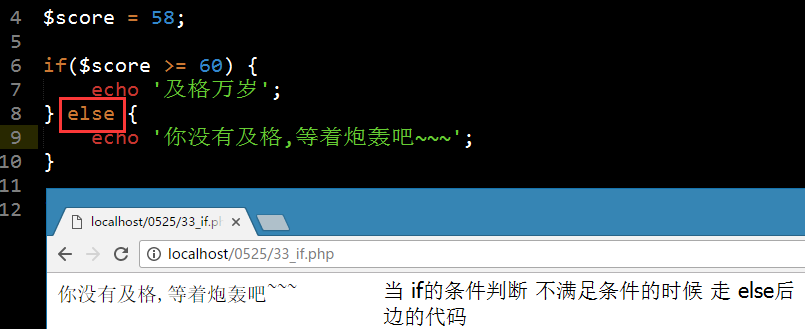
if(条件判断) {

如果条件判断为真,则执行大括号里的代码

}



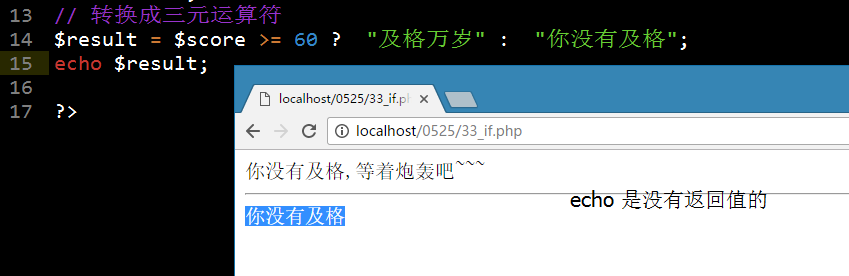
## if-else形式



if – else 的形式有些情况 可以转换成 三目(元)运算符

echo 函数没有返回值;

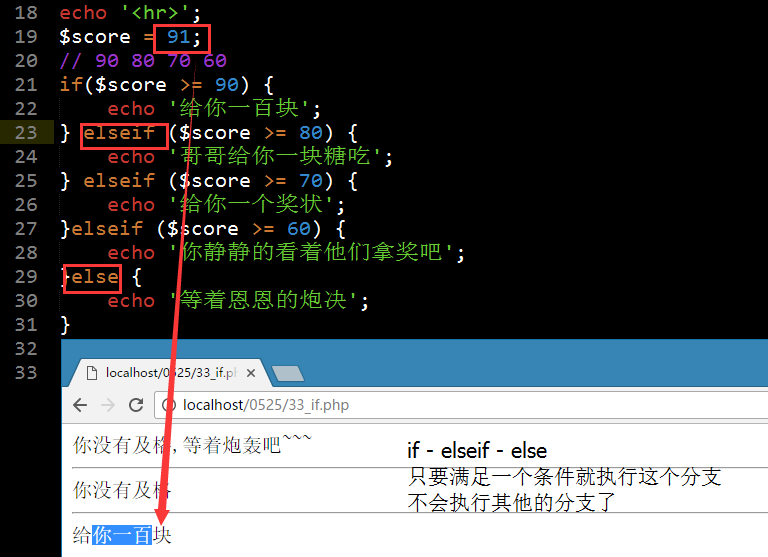




## if-elseif-else形式

多个条件判断我们用 if-elseif-else

只执行一个分支,执行完一个分支之后就跳出if了



php中 elseif 是在一起的.是连着的.

# 第34讲 switch语句

switch语句又叫做开关语句

## 语法形式

switch(表达式) {

case 表达式1: 语句

case 表达式2: 语句

case 表达式3: 语句

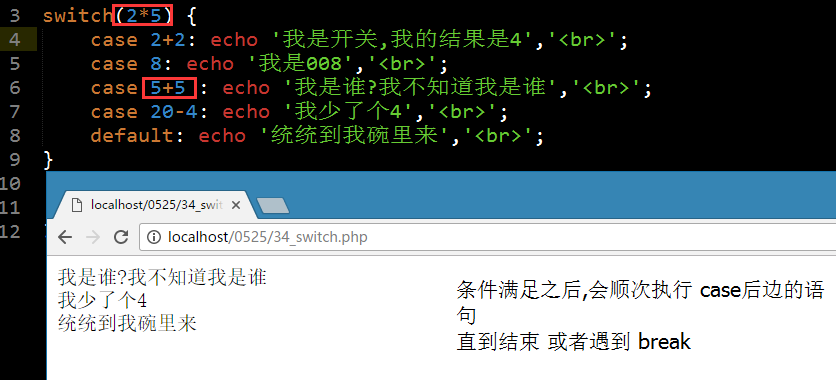
…

default: 语句

}

## 执行原理

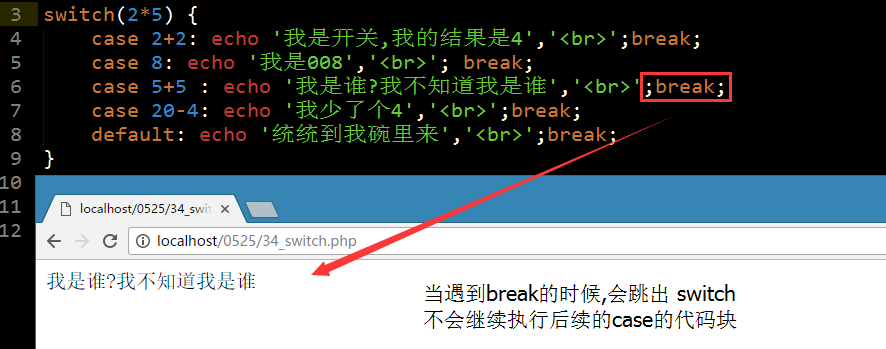
① 先计算 switch后边表达式的值,再计算case后边的值,然后 switch后边的值 分别和case后边的值进行比较, 当 2个值相当的时候,开关开启.条件满足之后会顺次执行case后边所有的语句,直到结束或者遇到break;



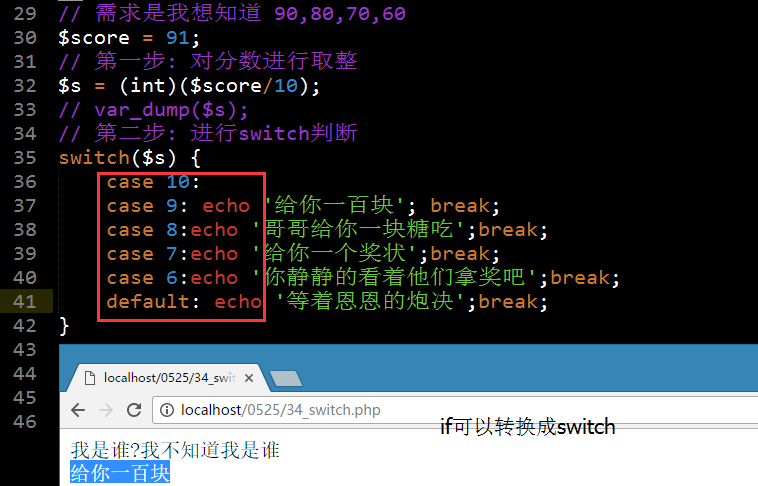
②当switch的表达式的值 和 case后边的值都不相等的时候,执行 default



③ 如果我们想 case后边只执行 当前的代码块,不执行后续的 case的代码块,我们需要在对应的代码后边加 break; break并不属于 switch的一部分,它属于for循环的一部分



## if与switch的区别



if 是一种条件判断

switch是一种状态 ,switch更多的是对一种已知的情况的所有的罗列

举例来说: 八个方向 ,这8个方向是确定的

性别: 男,女,保密

色子: 六个点

switch(sex) {

case ‘男’: echo

case ‘女’: echo

}’

# 第35讲 while语句

## 语法格式

while(条件判断) {

循环代码

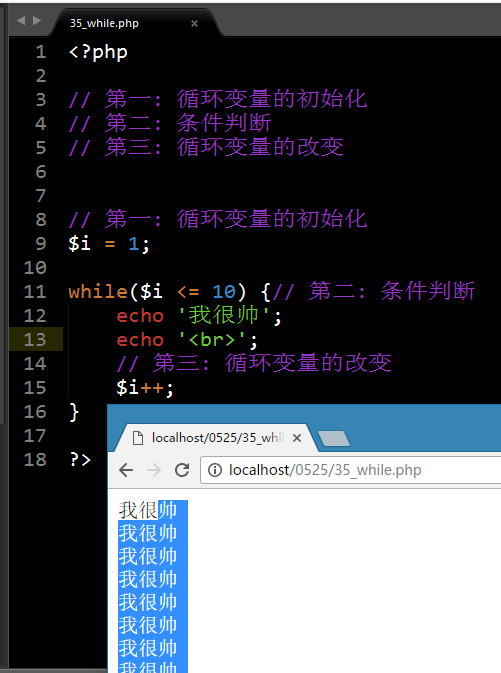
}

循环必须有 循环三要素.

第一: 循环变量的初始化

第二: 条件判断

第三: 循环变量的改变



## 执行原理(变量值的改变省略了)

① 先进行while条件判断,如果条件判断为假,则跳出while循环

② 如果条件判断为真,则执行while循环

③ 当while循环执行完一次之后,再重复执行第①步

## 流程图



# 第36讲 do-while语句

## 语法形式

变量的初始化

do {

循环代码

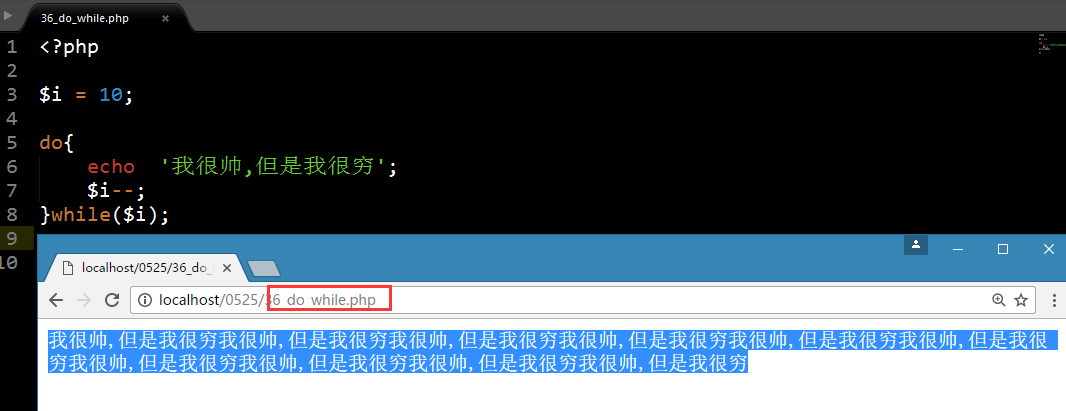
变量改变

}while(条件判断);

变量初始化

变量改变

条件判断



## 执行原理(不包括循环变量的改变)

①先执行一次循环体

②执行条件判断,如果条件判断为真,则 继续执行第①步

③ 当条件判断为假的时候,跳出do-while循环

## 流程图



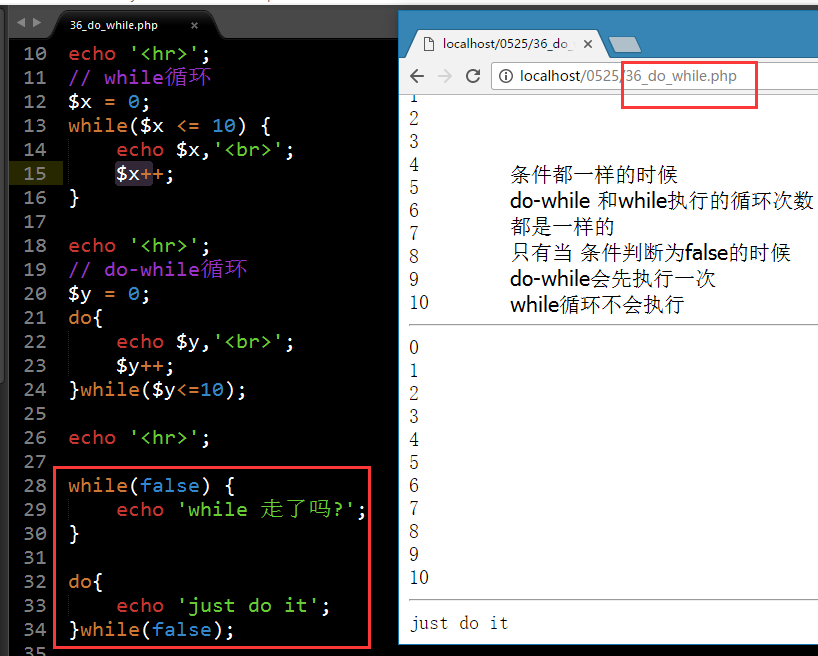
## while和do-while的区别

while 和 do-while 都是能够完成指定的循环;

当while和do-while的判断条件和循环变量都一致的时候,他们的循环次数是一样的.

只有当 条件判断为false的时候, do-while会先执行一次循环然后结束;

while循环一次也没有执行.



# 第37讲 for循环

## 语法形式

for(表达式1;表达式2;表达式3) {

循环体

}

表达式1:一般用于变量的初始化, 表达式1可以初始化多个变量,多个变量之间用逗号分隔.

表达式1只执行一次!!!!

表达式2: 条件判断

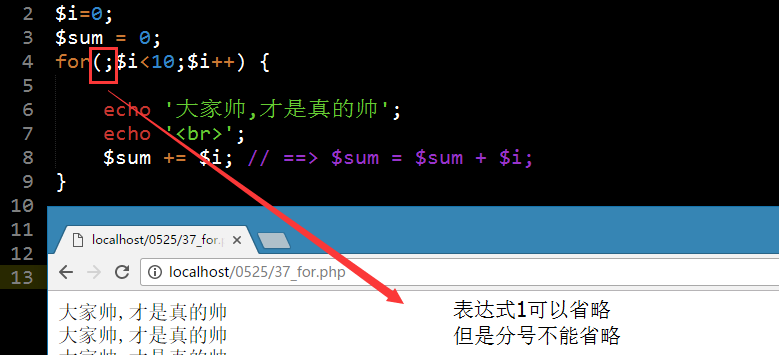
表达式3: 变量的改变

## 流程图

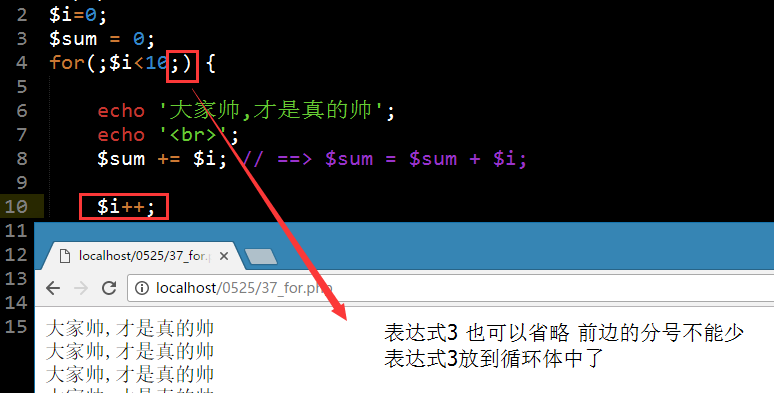


## 注意事项

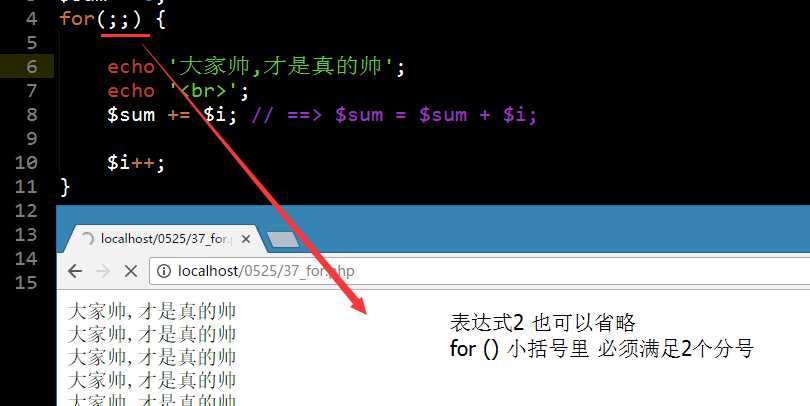
①表达式1可以省略,省略之后,表达式1后边的分号不能省略



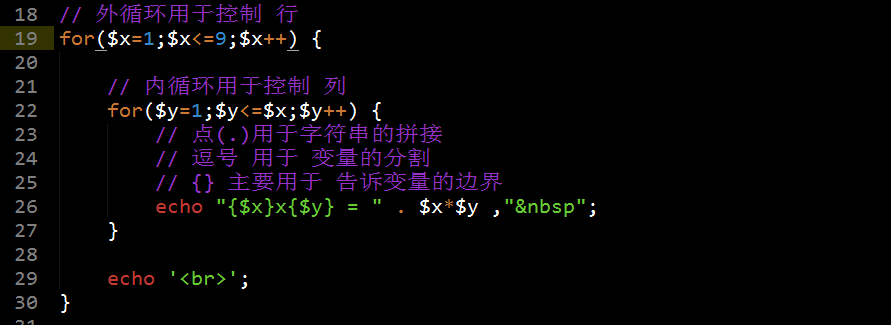
② 表达式3也可以省略,省略之后,表达式3前边的分号不能省略



③ 表达式2其实也可以省略,但是 成为了死循环,条件一直为真



用for循环实现一个 9 \* 9乘法口诀



# 第38讲 循环的中断语句

## Continue

一旦执行到了continue语句，就马上结束本次的当前循环（也就是说，后面剩余的循环体就不执行了），但是，整个循环并没有结束，而是重新开始了下一次循环！

至于整个循环结不结束，是由循环的条件决定的！

## Break

一旦，执行到了break语句，就马上结束当前循环的整个循环！

## 中断的层次

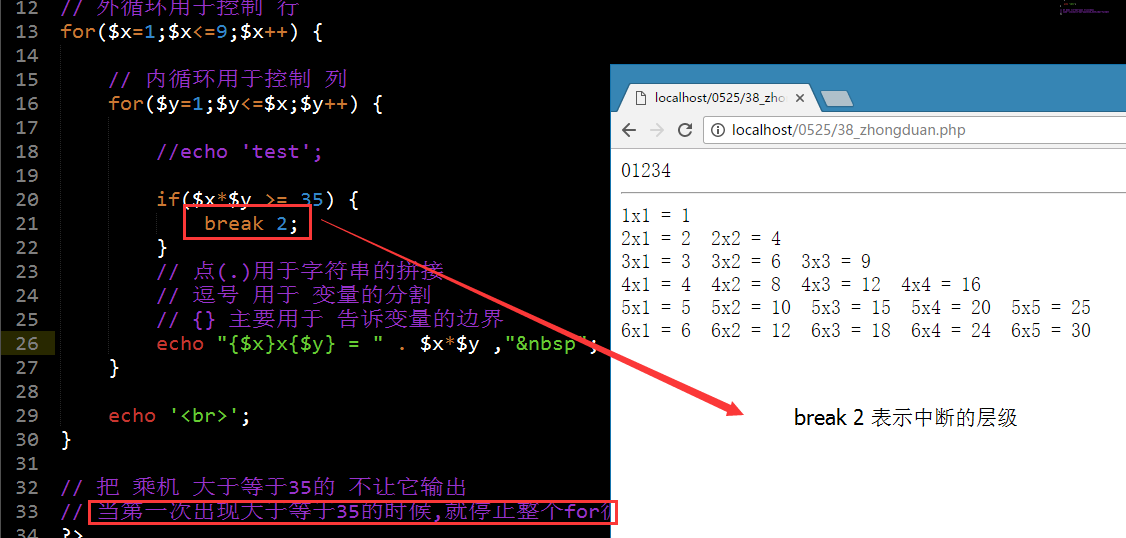
指的是从当前的中断语句开始算起，往外层循环的中断的层数，默认值是1；

break 1 中断当前循环，也是默认值

break 2 中断当前循环和上一层循环，也就是中断两层

类似的，还有：

break 3



# 第39讲 流程控制语句的替代语法

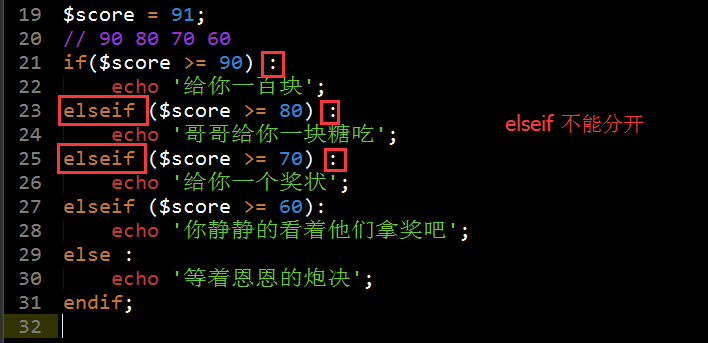
## 原因

PHP支持流程控制语句的替代语法。

当大量的HTML与PHP代码混编时，为了方便区分流程控制语句的开始及结束位置，可以使用流程控制的替代语法。

## 语法规则

* 将以前每个语句或每个子语句最开始的左大括号都用冒号来代替；
* 整个语句的结束（注意不是子语句的结束），使用另一种结束标签，比如：endif，endwhile，endfor，endforeach等，并在结尾加一个分号！
* do-while语句没有这种替代语法！



# 第40讲 文件载入

文件载入更多是体现了一个分层的思想

.当我们需要载入文件的时候,我们载入就可以了.

当需求发生改变的时候,我们只需要改变一个地方就达到了所有的地方都更改的情况

文件载入有4中形式:

include

include\_once once(一次就好)

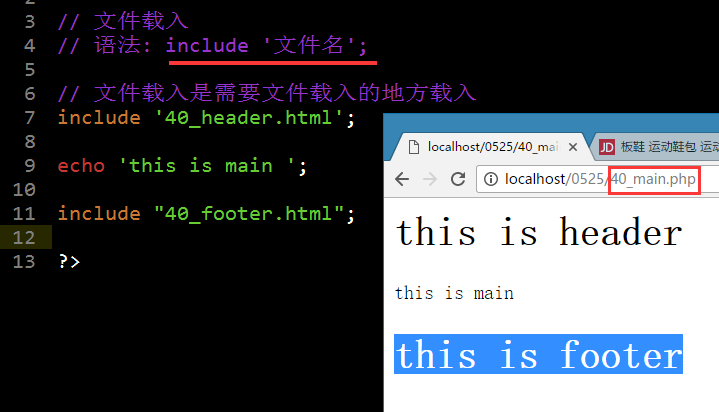
require 必须的

requie\_once

## include

### 语法格式

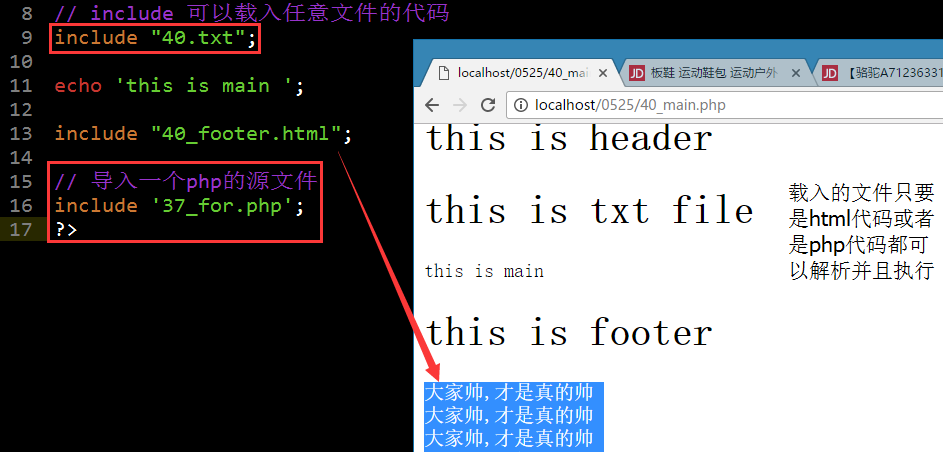
include “文件”;



### 注意事项

① include 可以载入任意文件的代码 . 但是一般我们就是在html/php

② 任意文件的代码只要是php或者是html, 都可以解析并且执行



### 深度剖析载入过程

.php文件 有2种模式:

php模式: php标签内为php模式

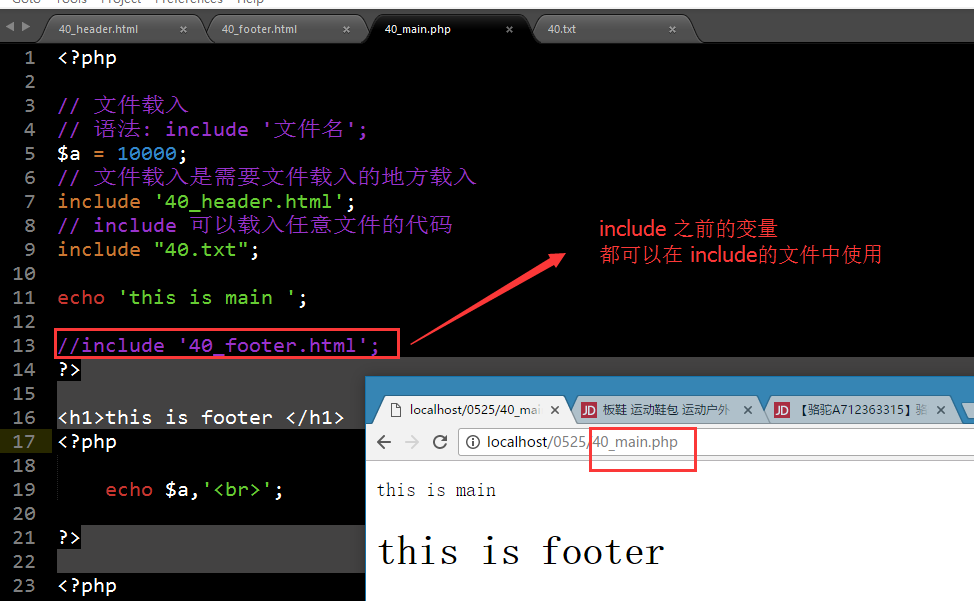
html模式: php标签外就是html模式



① 当 脚本遇到include之后,会退出php模式,就进入了html模式

② 就是把目标文件的代码相当于复制过来, 会进行解析并且执行

③ 结束html模式,进入到php模式



## 载入路径问题

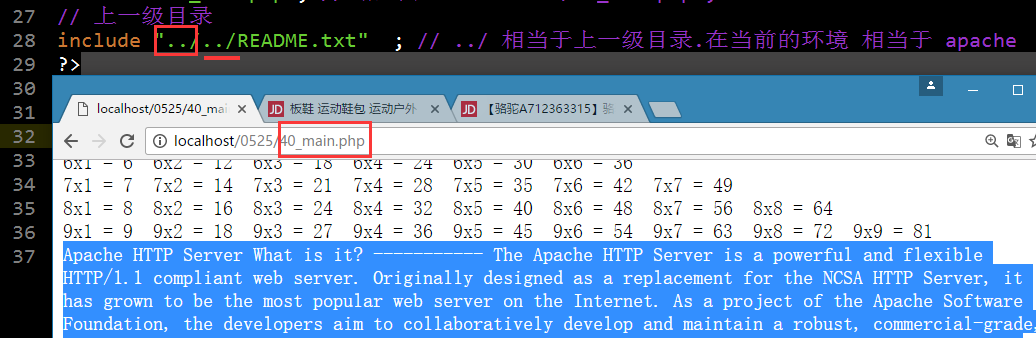
### 相对路径

相当于当期的文件夹

./ : 表示 相当于当前的文件夹

../: 表示 上一级的文件夹

../../ 表示更上一级



### 绝对路径

绝对路径就是从盘符开始

c:\windows\system32\drivers\etc\hosts

域名解析文件

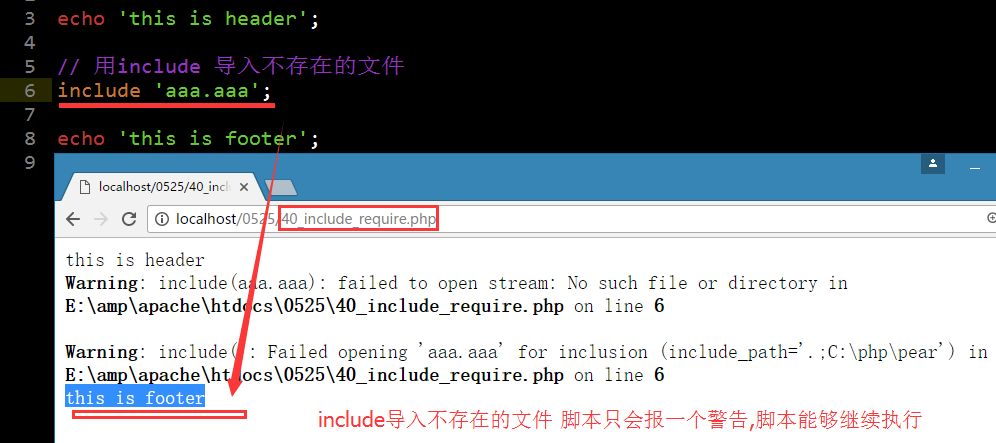
## 不同载入方法的区别

include require include\_once 和require\_once 都可以文件载入

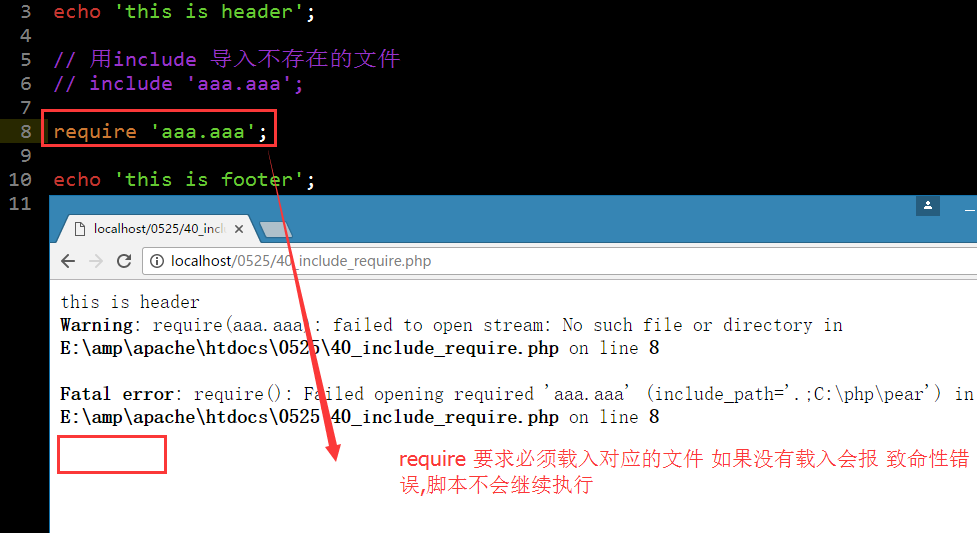
### require和include

require 和include 的区别 同样适用于 require\_once 和 include\_once

当我们载入不存在的文件的时候,include 不会报错,会报警告,脚本能够继续执行;



当我们载入的文件不存在的时候,require 会报错误,脚本停止执行



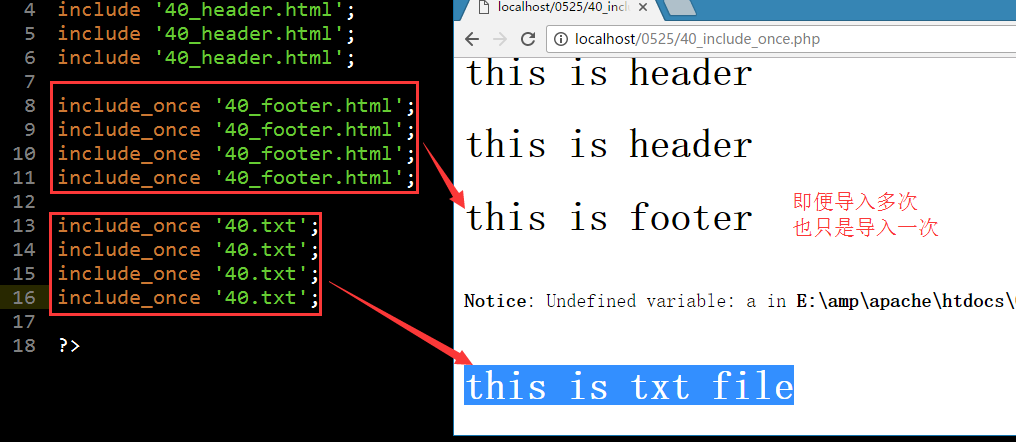
### include和include\_once

incldue 和 include\_once 的区别 同样适用于 require 和require\_once

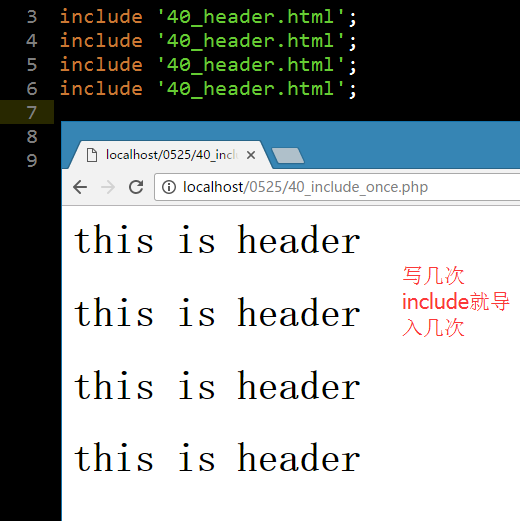
once 一次就好.

include\_once ‘文件名’;

include\_once 会加载相同的文件名只会加载一次!!!



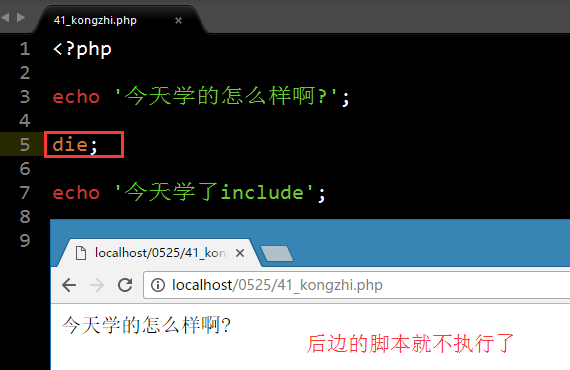
include



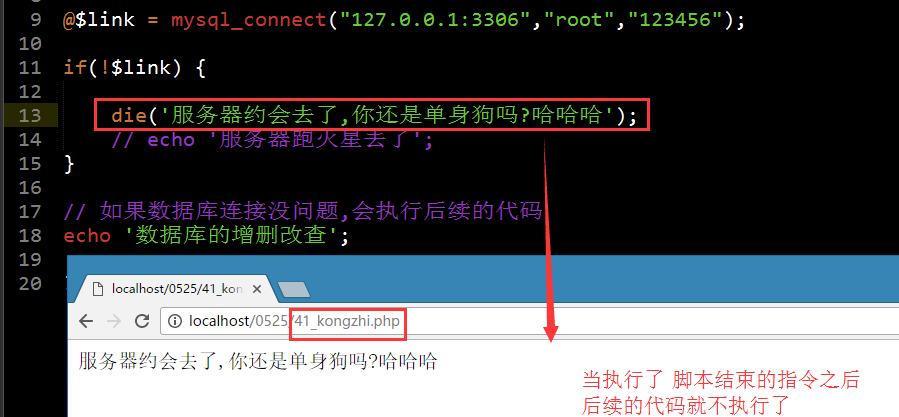
# 第41讲 脚本的执行控制

## 终止脚本执行

die 和 exit 都可以终止脚本的执行 (不用去比较这2个);



die一般在连接数据库的时候,如果 连接数据库出现了问题,就停止



die/exit 有两种调用形式:

die/exit; 这种形式没有任何提示语

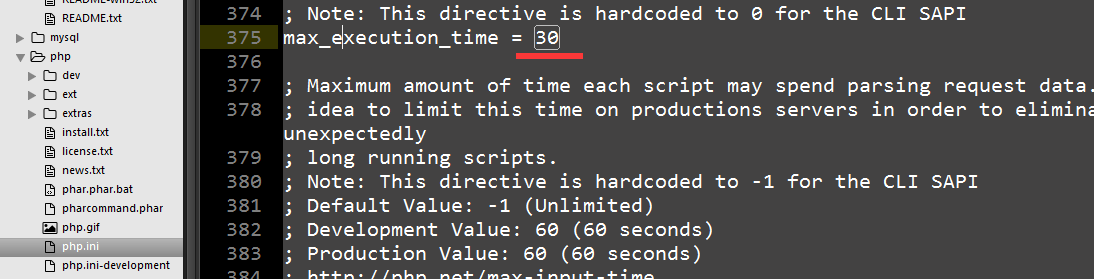
die(提示语)/exit(提示语); 这种形式有提示语可以输出

## 脚本延迟执行

sleep(秒数)

sleep 可以让当前脚本延迟几秒执行. 在我们开发过程中可以查看一些临时的变量的信息

千万不要让脚本睡眠时间超过 30秒



# 第42讲 错误处理

## 错误的分类

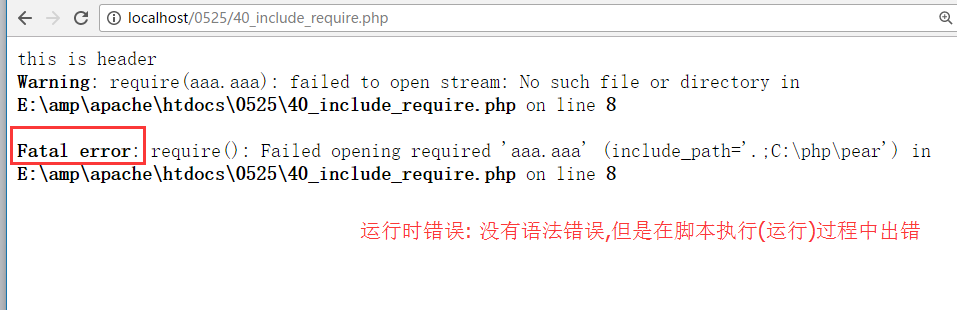
### 语法错误（解析错误，parse error）

解析语句时，由于语句不合乎代码规范，因此发生解析错误，后续代码不执行。



### 运行时错误（run-time errors）

程序语法没有错误，但是出现未定义变量等情况。



### 逻辑错误

用户自行指定的错误，与预期的数据不一致等情况，如数据不合法。

## 错误的分级

### 系统错误

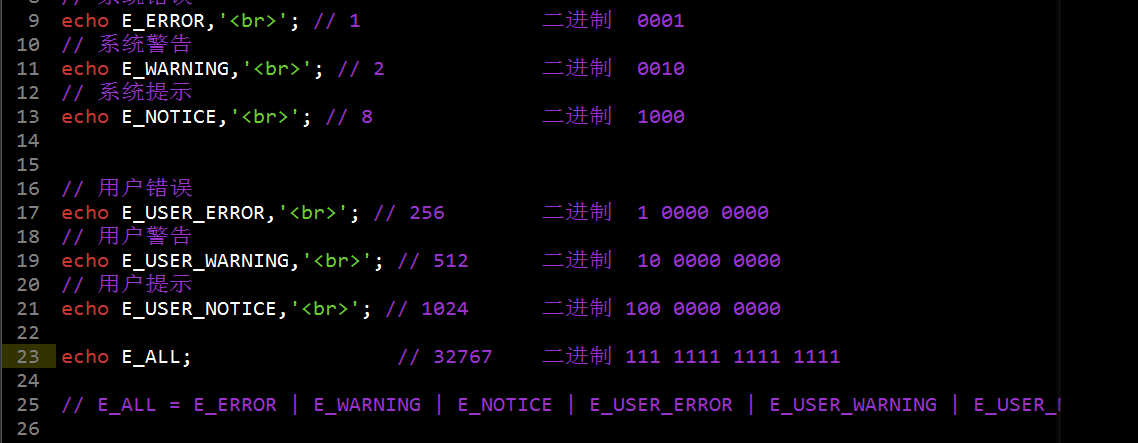
* **E\_NOTICE**: 通知性错误,是最小的错误，当发生“通知性错误”时，会弹出一个提示信息。它不会中断程序继续向下运行。如：使用未定义变量或者常量时。 有时间解决就行
* **E\_WARNING**: 警告性错误,warning，当发生“警告性错误”后，会弹出一个警告信息，不会中断程序向下运行。如：使用include包含了一个不存在的文件等。 当天能够解决就行
* **E\_ERROR：**致命错误，fatal error,当发生“致命性错误”后，会弹出一个致命错误信息，并中断程序向下运行。如：使用require包含了一个不存在的文件等。致命性错误需要我们立刻马上解决

### 用户错误

* **E\_USER\_NOTICE**
* **E\_USER\_WARNING**
* **E\_USER\_ERROR**

### 其他

* **E\_ALL：**所有的错误及警告



## 错误的触发

### 正常触发

除了用户错误以外，其它错误都是系统触发的

### 人工触发

手动触发相应的错误程序，一般为用户级别的错误。

触发函数

trigger\_error(错误提示信息，错误级别);

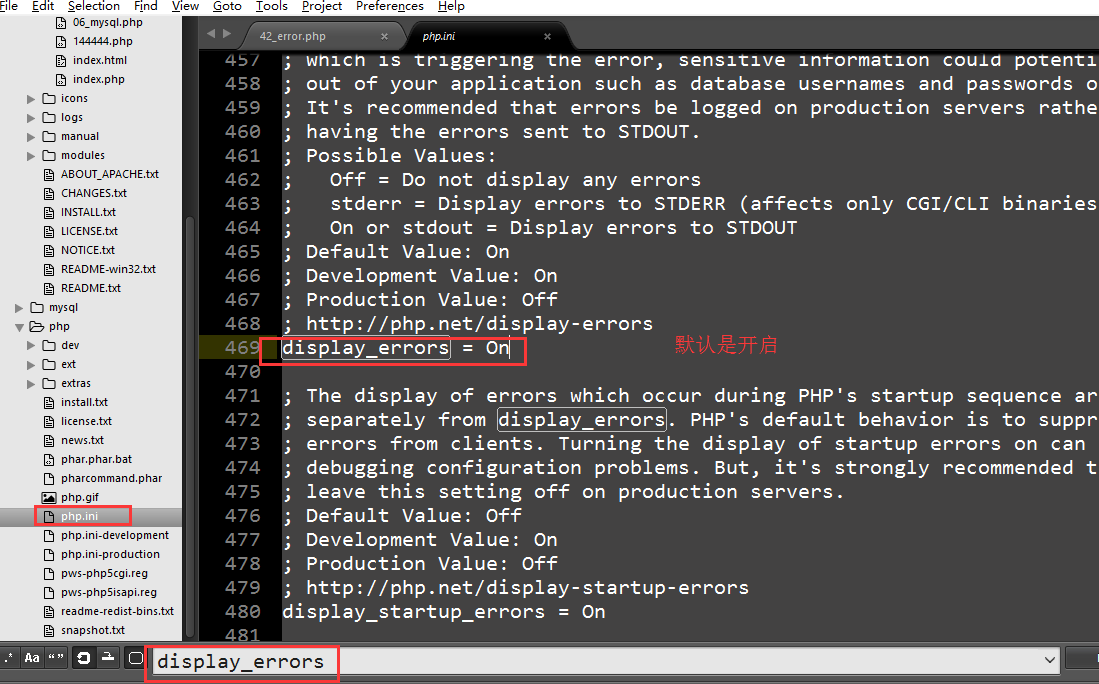


## 错误的显示控制

### 错误信息显示控制

* 在php.ini中,display\_errors=on;// on表示显示,off表示不显示

当我们想让所有的脚本都显示或者不显示,我们就需要修改php.ini

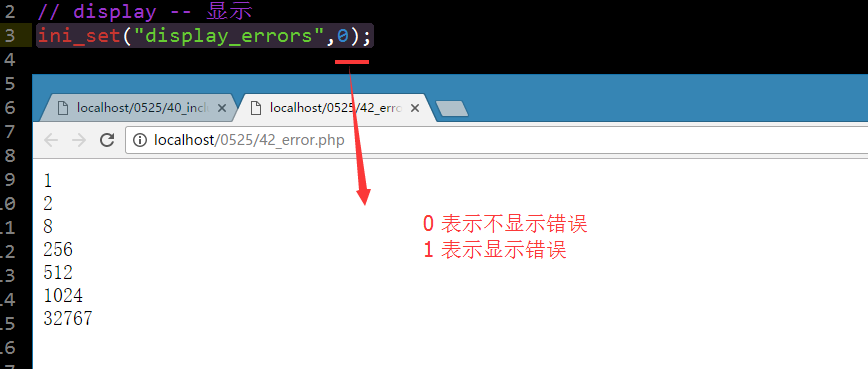


当修改成为off之后, 就不显示了

重启apache

* 脚本中,ini\_set(“display\_errors”,1)//1表示显示,0表示不显示,也可以是true/false

我们只需要在当前的脚本中生效,只需要在脚本的顶端,添加设置



### 显示错误信息的级别

* 在php.ini中,error\_reporting = E\_NOTICE | E\_WARNING | E\_ERROR
* 在脚本中,ini\_set(“error\_reporting”, E\_NOTICE | E\_WARNING);

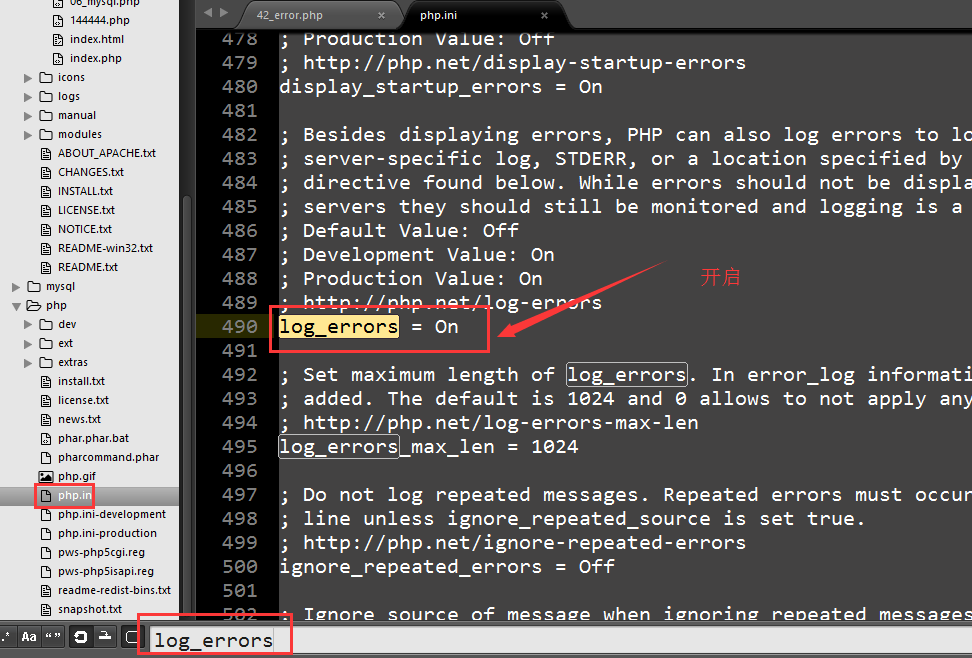
课下的时候可以去试试,一般我们都会记录所有的错误

## 记录错误日志

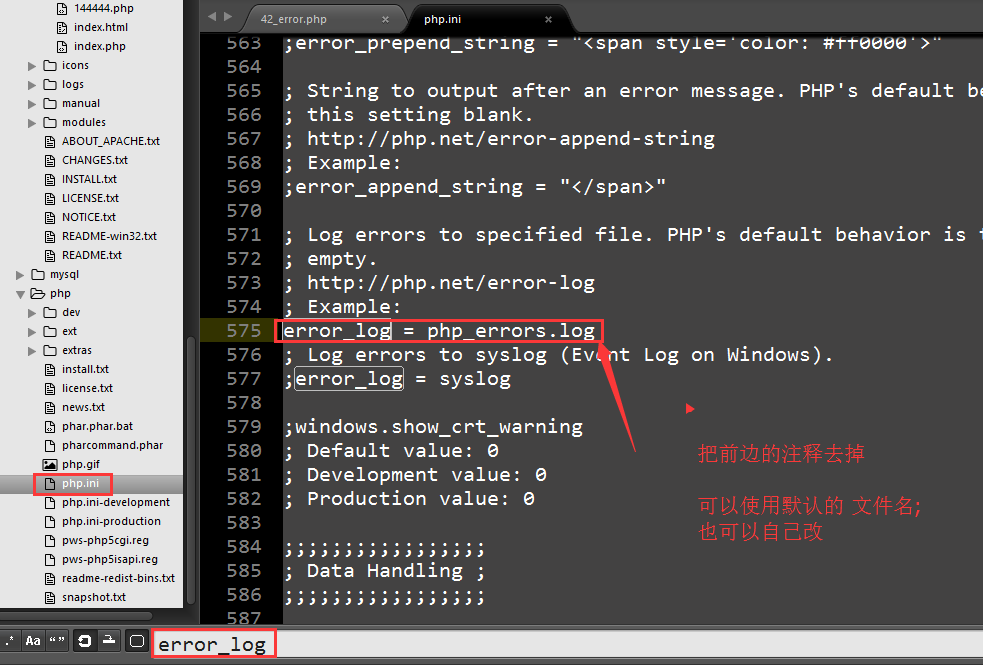
记录日志有2中途径:

### 在php.ini中设定

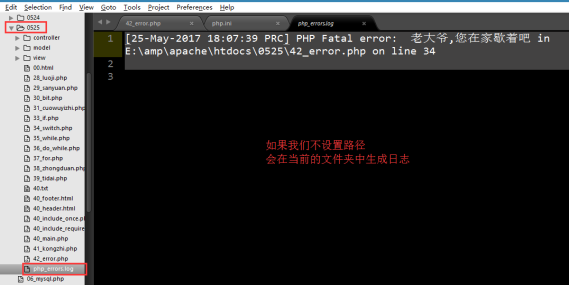
log\_errors = on;//on表示记录,off表示不记录 这个是前提



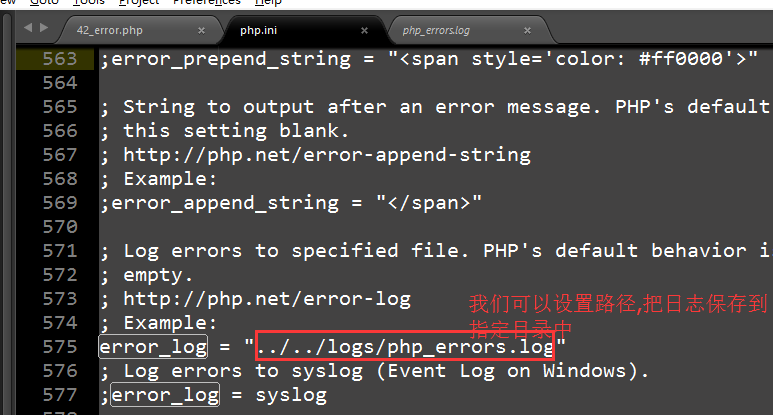
error\_log = “错误日志名”;//设定错误日志的文件名



在脚本文件的当前文件夹中会生成日志文件



在当前脚本文件夹中生成日志文件不太好,我们可以设置路径,把日志全部写入到apache的指定目录里



如果该文件没有指定路径,则系统会在每个文件夹下建立该文件并记录进去.

我们还可以设置error\_log = syslog; // 此时不会记录错误到日志文件,会把错误写入到系统的错误日志

### 在当前的脚本中设置

ini\_set(“log\_error”,On);

ini\_set(“error\_log”,”err.log”);

ini\_set(“error\_log”,syslog);//记录到系统的日志

## 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。

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);