二进制：

计算机所有的信息的存储，运算，传输都是使用二进制进行。

单位：

位 bit 二进制中的1个数位

字节 byte 八位二进制数

k字段 kb 1024\*1byte

m 1024kb

g 1024m

t 1024g

p 1024t

ascii 1字符占据8位二进制数，占据1个字节

gb2312

gbk

big5

gb18030

utf8 世界词典

ascii 1个字符占据1个字节

gbk 1个字符占据2个字节

utf8 1个字符占据3个字节

运算符：

逻辑短路：

逻辑与短路

逻辑或短路

and \ or 小于 = 小于 &&\||

位运算

&

|

^

~

<<

>>

@

ini\_set \ ini\_get

php.ini

php.exe

php文件的访问是通过浏览器，访问apache，才能运行一个Php文件。

通过浏览器在不需要创建apache服务器的情况下：php提供了一个内置的web服务器。

php.exe –S 域名:8000

不需要apache也不需要浏览器，运行一个php文件：

php.exe -f 文件名

php.exe --ini

php.exe -m

php.exe –r “一条语句”

一、流程控制

计算机中的流程控制可分为三种：

顺序结构

程序自上而下的一个执行过程。

分支结构

根据某一条件将程序转向不同的分支处执行。

循环

计算机最大的特点计算机速度快，计算机主要用于解决重复性的问题。

二、分支结构

1、if分支

①、单分支

语法：

if(表达式){

//语句体

}

说明：

当表达式成功时，执行语句，如果不成功直接结束if结构，继续其后的语句执行。



②、双分支

语法：

if(表达式A){

语句体A

}else if(表达式Ｂ){

语句体B

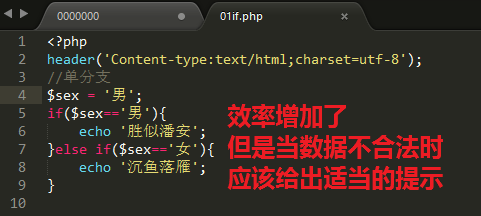
}

说明：

当表达式A成功时，执行语句体A，如果不成立来判断表达式B，如果表达式B执行语句体B

当任何一个语句体执行结束后，不再会去进行其他表达式的判断。

示例：



③、多分支

语法：

if(表达式A){

语句体A

}else if(表达式B){

语句体B

}else if(表达式C){

语句体C

}else{

缺省语句体

}

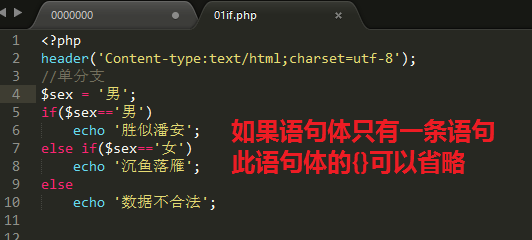
说明：

自上而下的判断各个表达式，当某个表达式成立时，执行相应的语句体

语句体执行结束后，不会再判断其他的表达式。

当所有的表达式都不成立，执行缺省语句体。





2、switch结构

当一个问题需要判断的条件比较多时，Php又提供了一个switch分支结构。

语法：

switch(变量){

case 值1:

语句体1

break;

case 值2:

语句体2

break;

case 值3:

语句体3

break;

default:

缺省语句体

}

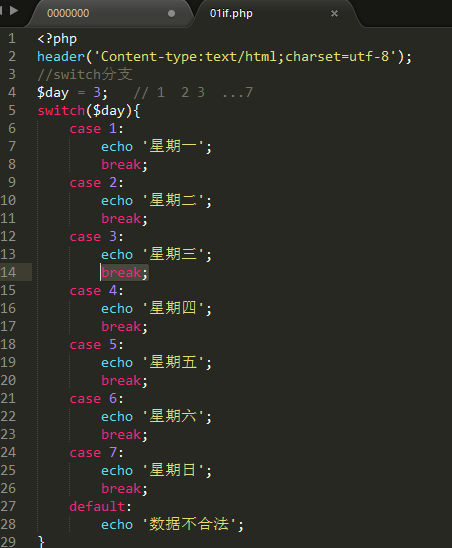
说明：

根据switch后的变量与case后的值的匹配情况，将程序转向不同的语句体执行。

break可以省略。

当一个语句体执行结束后，如果没有遇到break，则会执行其后的语句体，不会判断其后case的值。

示例：



if与switch的区别：

if结构的条件可以表示一个范围。

switch结构主要用于表示一个具体的值。

三、循环结构

1、for循环

循环结构主要是用于解决重复性的问题。

语法：

for(循环控制变量初始化;表达式;循环控制变量的更改){

//循环体

}

说明：

循环控制变量初始化;表达式;循环控制变量的更改 循环3要素。

a、首先执行循环控制变量初始化，此步仅执行1次。

b、判断表达式是否成立，如果不成立直接结束循环体，继续其后的语句执行。

c、如果成立，执行循环体。

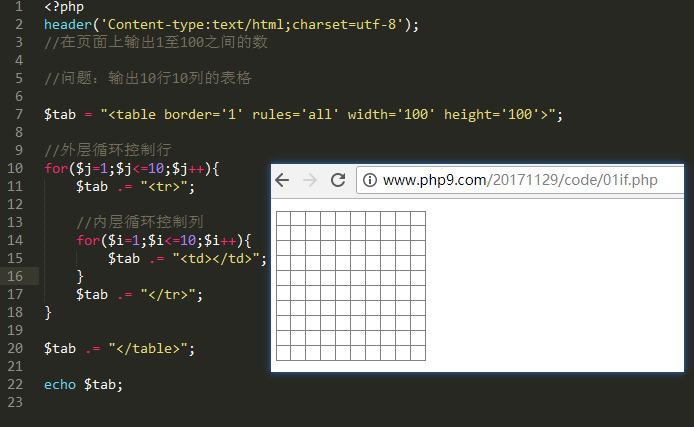
d、循环体执行结束后，执行循环控制变量的更改，

e、往复执b、c、d，直到循环控制变量的值主表达式不成立时，结束循环结构。

对于for循环结构，要求大家能够作到以下几点：

* 能够使用for循环，模拟任意次数的循环。
* 能够使用循环控制变量来模拟数据或规则

循环嵌套：



提示：

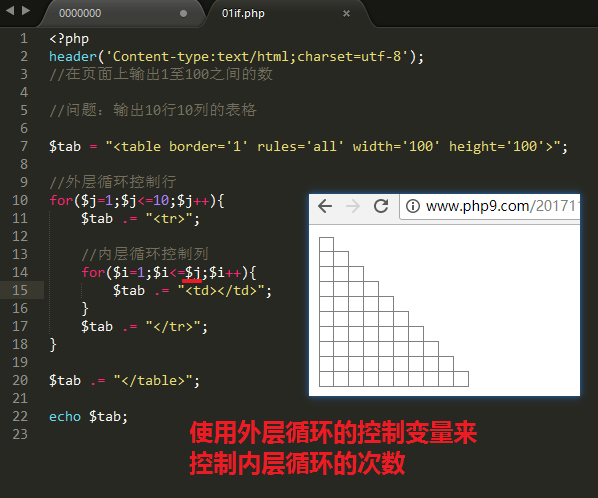
双层循环的嵌套主要解决具有行列特性的问题。

使用程序语言解决问题的思路：

首先要判断使用何种流程控制。

再将问题中的条件转换为计算机能够表达的式子。

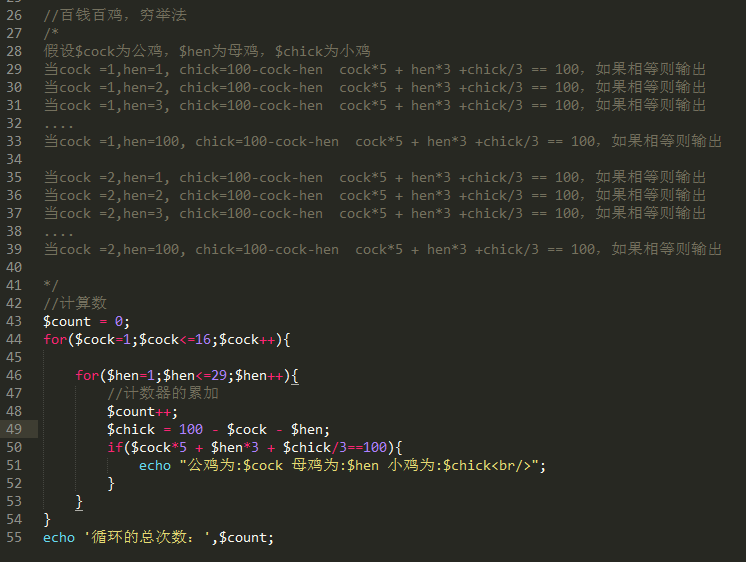
示例：



课堂案例：

百钱百鸡，100文钱买100只鸡，公鸡5文1只，母鸡3文1只，小鸡1文3只。

穷举法：



2、while循环

语法：

while(表达式){

//循环体

}

说明：

只要是循环必须有3要素组成。但是while循环的语句结构中仅出现了表达式。

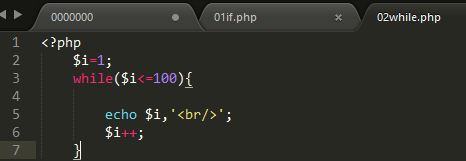
另外两个要素需要自己设计。

首先判断表达式是否成立？

不成立，直接结束循环结构，继续其后的语句执行。

成立，执行循环体，循环体执行结束后，再来判断表达式。循环往复的执行。直接到表达式不成立。

示例：



for与while的区别

* 对于for循环结构，主要用于循环次数已知或根据某一条件可以确定。
* while用于循环次数未知的循环。

3、do..while

语法：

do{

//循环体

}while(表达式)

说明：

首先执行循环体

循环体执行结束后，判断表达式是否成立，

如果不成立，直接结束循环结构，继续其后的语句执行，

如果成立，再执行循环体。往复的执行，直到表达式不成立。

示例：



4、循环的结束与退出

①、continue

语法：

continue 【n】;

说明：

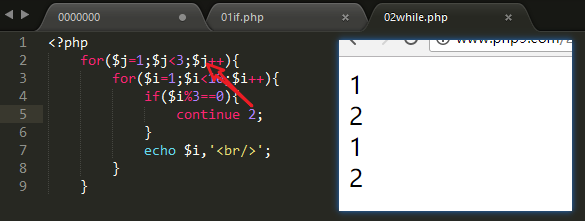
n的取值是一个整数，如果省略默认是1，

n主要是用在循环嵌套的情况下。

结束当前循环结构的本次循环，继续上n层循环结构的下一次循环。

示例：





②、break

语法：

break 【n】;

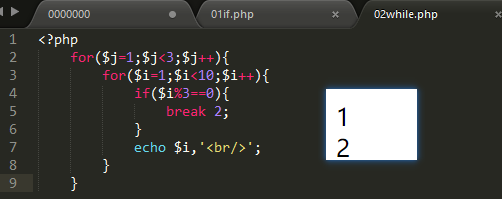
说明：

n的取值是一个整数，如果省略默认是1，

n主要是用在循环嵌套的情况下。

直接结束上n循环循环结构

示例：



四、流程控制语句的标签语法

php在将php代码嵌入到html文档时，又提供了一种标签语法。



①、if标签语法：

标准语法：

<?php if(...):?>

<?php endif?>

简化语法：

<?php if(...){?>

<?php }?>

②、for标签语法

标准语法：

<?php for():?>

<?php endfor?>

简化语法：

<?php for(){?>

<?php }?>

③、while的标签语法

标准语法：

<?php while():?>

<?php endwhile?>

简化语法

<?php while(){?>

<?php }?>

五、文件载入

1、介绍

在实际开发过程中，经常会在一个php文件中引入其他的文件。

被引入的文件可以是html也可以是php文件。

2、语法

①、require

require(文件名)

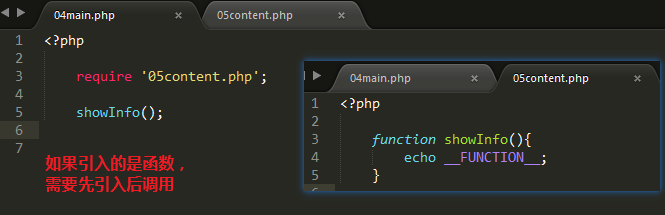
require\_once(文件名)

②、include

include(文件名)

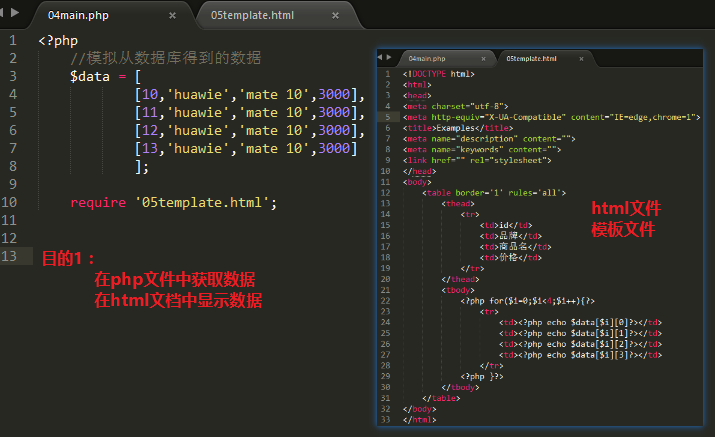
include\_once(文件名)

示例：

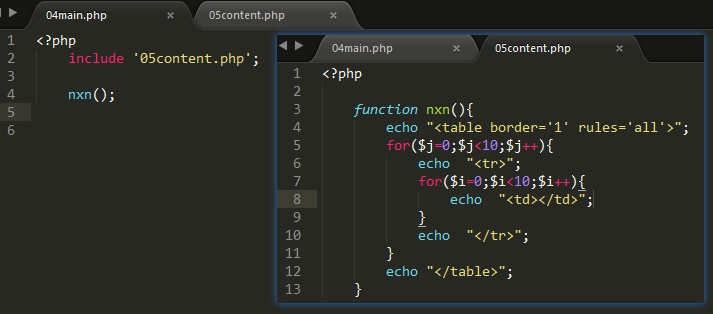


在引用文件时，主要有两种使用目的：

当前文件(php)有数据，想在被引用的文件(html)中显示。



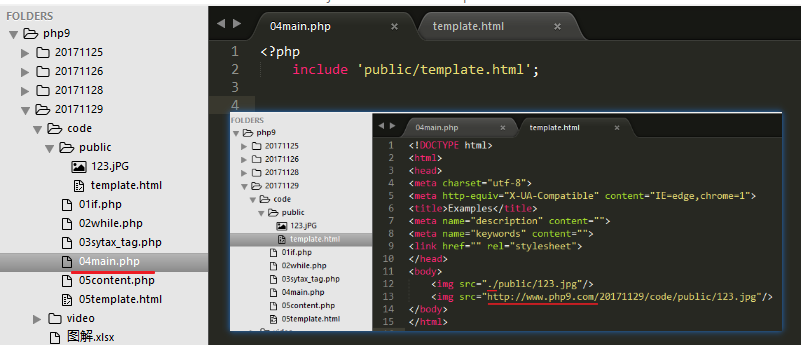
在当前文件(php)中想使用，另一个文件(php)中的功能性代码:



3、引入路径的问题

在实现项目中，对于html文件，我是不允许用户直接请求，而是指向一个php文件，让php文件来引处这个html文件。

当一个php文件引入一个html文档时，html文件本身也会引入一些其他的文件，如：图片文件、css文件、js文件。这时会发生路径更改的问题。路径要以当前php文件所在的路径为起点。





4、include与require区别

include 在引入文件时，如果被引入的文件不存在则会报错，但程序还会继续向下执行。

require在引入文件时，如果被引入的文件不存在，则会上断程序的执行。

经验法则：

require一般用于引入php文件。因为php里面一般书写的是功能性的代码。

include一般用于引入html文档。

include、require VS include\_conce、require\_once的区别

include\_once、require\_once每次在引入文件时，都会检查所要引入的文件之前有没有被引入过，如果有引入过就不会再引入。

include、require没有这样的检查过程。

5、\_\_DIR\_\_\\_\_FILE\_\_

\_\_FILE\_\_ 用于获取文件所在的完整文件名。

\_\_DIR\_\_ 用于获取文件所在的路径。

\_\_FILE\_\_与\_\_DIR\_\_并不会被引入后所更改。永远代码\_\_FILE\_\_与\_\_DIR\_\_所在的文件的文件名与文件夹名。

六、错误处理(了解)

1、错误的分类

①、编译错误

在编译的过程中发生的错误就编译错误，编码错误是最容易解决的。很多是由于书写错误。



②、执行错误

在编译通过后，在执行阶段发生的错误。此种错误一旦发生，会根据错误的等级，来决定是否中断程序的执行。

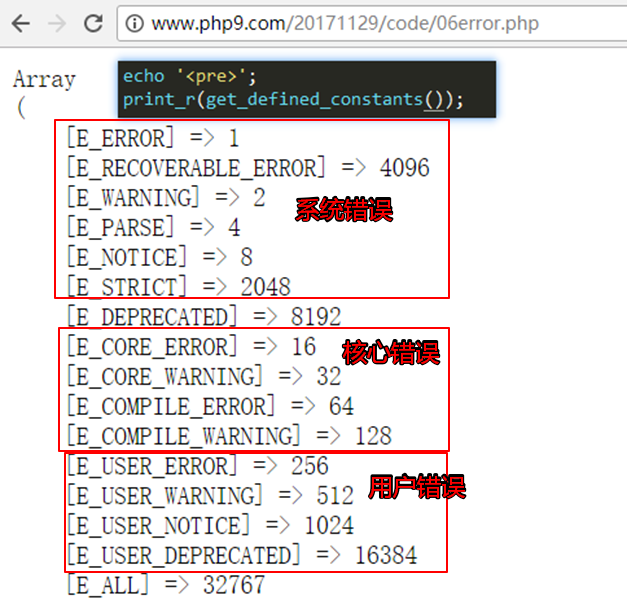
③、逻辑错误

由于程序的逻辑不严谨，而产生的错误。此种错误是最难排查的。程序可以正常执行，但最终的结果并不是我们所期望的。

2、错误代码

在php的错误处理机制中，每一种错误都使用了一个错误标识，错误标识是常量的形式存在。

①、获取错误代码



②、系统错误

E\_ERROR 致命错误，一旦发生就会中断程序的执行

E\_WRRING 警告错误，一旦发生并不会中断程序的执行，

E\_NOTICE 提示错误，一旦发生并不会中断程序的执行，

E\_PARSE 编译错误，一旦发生就会中断程序的执行

③、自定义错误

E\_USER\_ERROR 用户级的致命错误，一旦发生就会中断程序的执行

E\_USER\_WRRING 用户级的警告错误，一旦发生并不会中断程序的执行，

E\_USER\_NOTICE 用户级的提示错误，一旦发生并不会中断程序的执行，

④、所有的错误

E\_ALL

3、错误的触发

①、系统错误的触发

程序员无法干涉系统错误的触发，系统错误由php自已检测并触发的。



②、自定义的错误

trigger\_error();

语法：

trigger\_error(msg,type);

说明：

msg 错误描述信息

type 自定义错误的代码(E\_USER\_ERROR、E\_USER\_WARNING、E\_USER\_NOTICE)

示例：





提示：

使用错误处理机制的好处是，可以将错误记录起来。默认会被记录到apache的/logs/error.log文件中。

4、错误的控制设置

php错误控制可以在php.ini中进行。我们可以控制错误是否显示，还可以控制显示哪一类的错误。

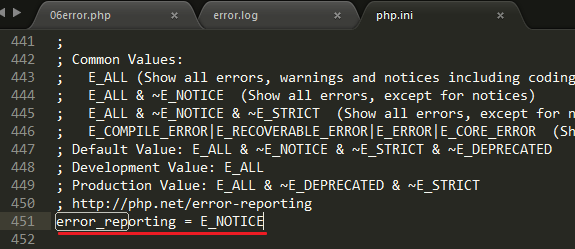
①、控制是否显示错误

display\_errors=on 显示错误

display\_erros =off 关闭错误的显示

②、控制显示哪一类错误

error\_reporting =



5、错误的日志文件的设置

我们还可以控制是否记录错误，以及错误日志的记录位置。

①、是否记录记录

log\_errors=on 记录错误

log\_errors=off 不记录错误

②、错误记录的位置

如果没有设置error\_log默认是记录到apache的错误日志中\logs\error.log

error\_log=syslog 会记录到操作系统的日志中

error\_log=文件名 记录到自定义的位置

