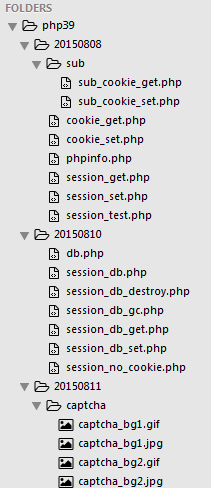
# 目录操作

## 递归获取目录内容

将某个目录内包括其后代目录内全部内容读取到！



通过函数的递归调用完成：

当解决问题，发现问题可以拆解成小问题，小问题解决方案与大问题一致！此时需要递归调用！

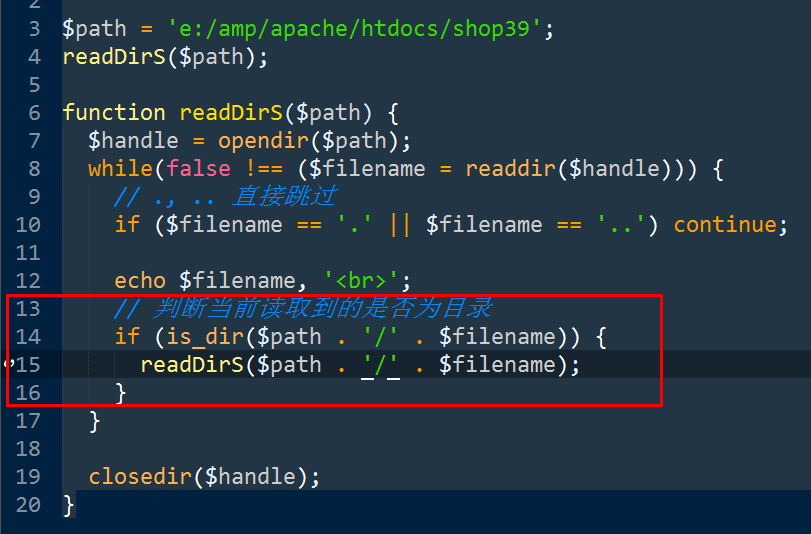
递归点：何时发生递归，读取到文件为目录时。

先读取某个目录内容（不包含子目录的）：

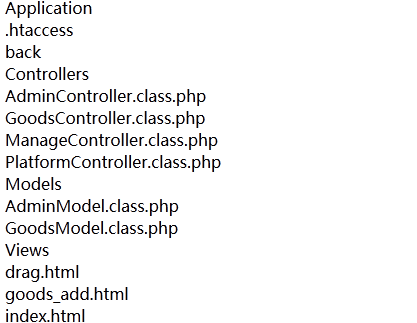


判断每个读到的文件是否为目录：

如果为目录，递归调用，将当前子目录作为参数传递。

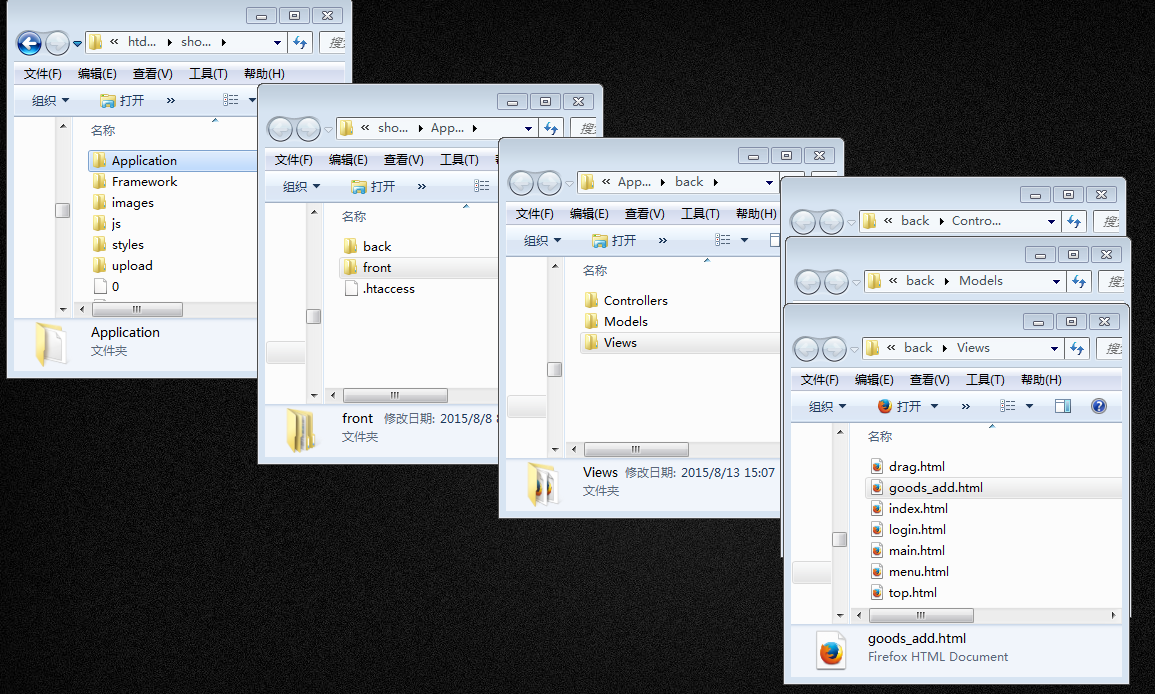


此时已经可以将所有内容读取：



递归出口在哪里？

当目录下没有子目录时，递归调用结束，返回调用他的函数（上一级调用）



### 树状展示结果

通过缩进展示，层级，级别，哪个文件时哪个文件的子文件！

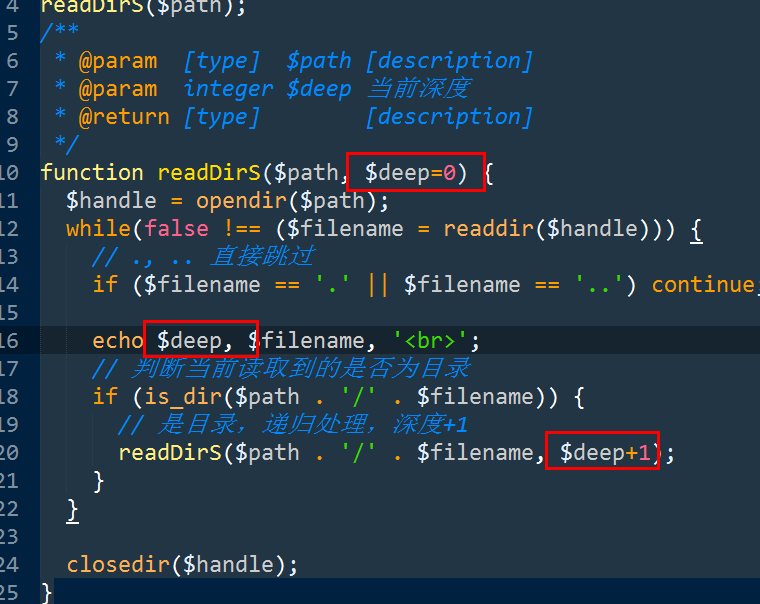
只需要确定缩进级别即可！

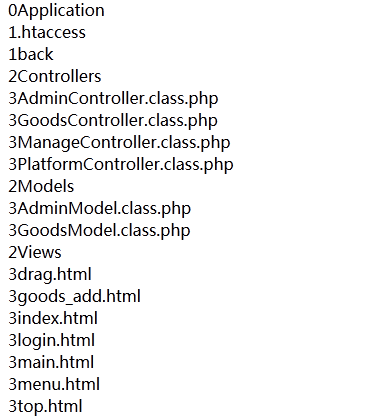
每当递归调用一次，则缩进级别增加1级。

递归调用深度，就是该函数确定的文件的缩进级别！

递归调用一次+1即可！

语法实现：

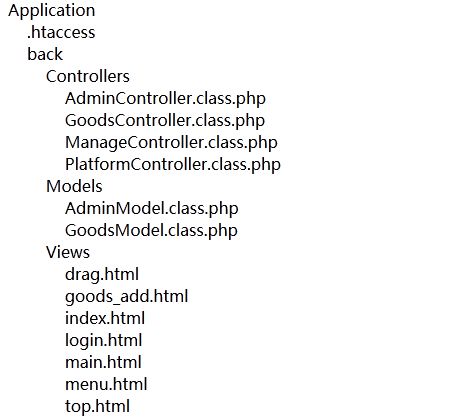




将缩进以缩进符表示：

Str\_repeat(字符串， N);





### 记录结果

需要记录每个文件名，缩进级别，[是目录还是文件]

Array(

Array(‘deep’=>0, ‘filename’=>’application’),

Array(‘deep’=>1, ‘filename’=>’back‘),

Array(‘deep’=>1, ‘filename’=>’controllers‘[, ‘type’=’dir’]),

编码实现：



额外：

将 类型 也 添加上！

## 递归删除目录

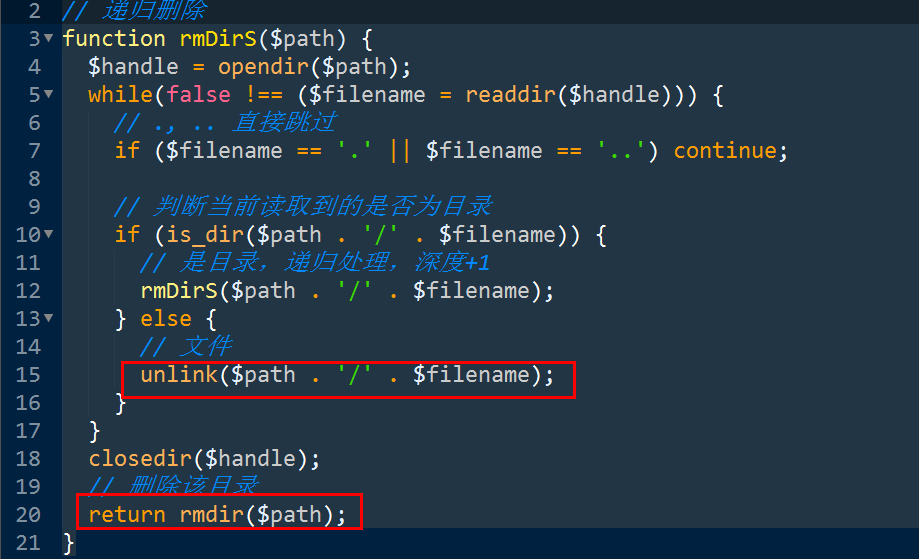
Rmdir()只能删除空目录！

当删除一个目录时，将目录内容删除掉之后，再删除目录本身。

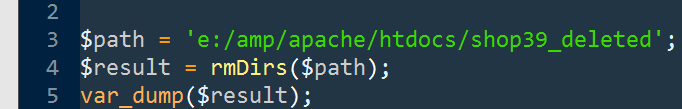
文件：unlink(文件地址)

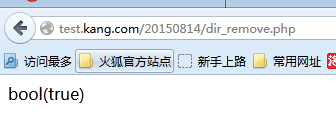
目录：还是递归！

类似于 递归获取的实现：



测试：





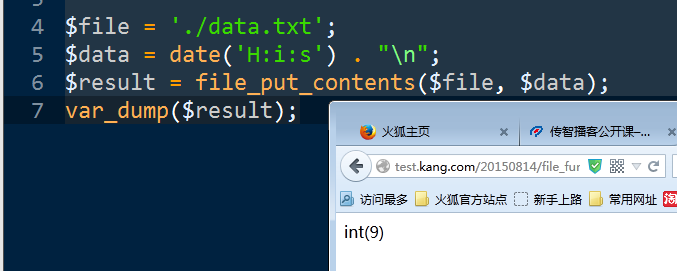
# 文件操作

## 读写基本操作

读写。文件作为内容（数据）容器！

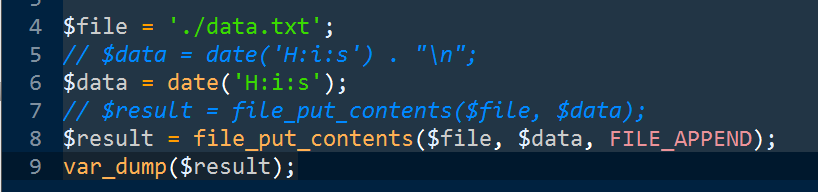
写入长度（字节） = File\_put\_contents(文件地址, 内容);

将内容写入文件！



默认为替换写，将原内容清空，再写入！

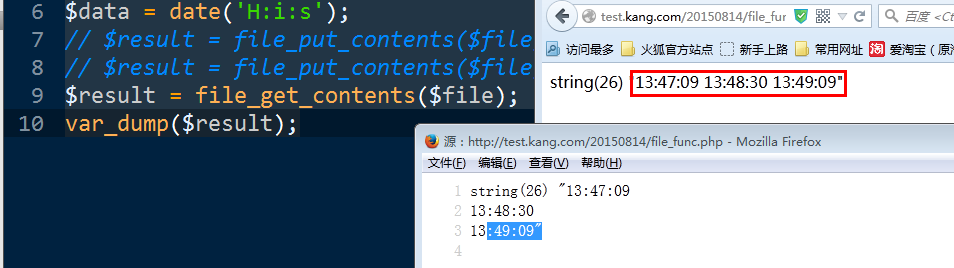
使用第三个参数FILE\_APPEND表示追加写：



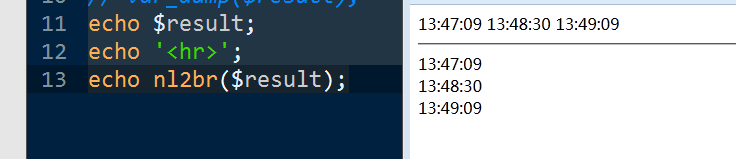
文件内容 = File\_get\_contents(文件地址);

将内容从文件中读取！

提示浏览器不解析\n换行的！



函数 nl2br(); 将换行符转换为br



## Unlink()

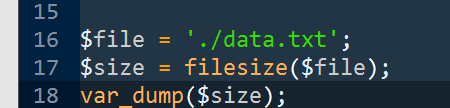
删除文件

## Rename()

文件移动（重命名）

## 大小int = Filesize(文件地址)

文件大小



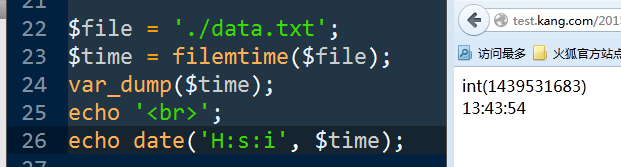
## 布尔 = File\_exists(文件地址);

文件是否存在

## 时间戳 = Filemtime(文件地址)

Modify

文件的最后修改时间！



## 文件句柄读写操作

基本单位都是字节：

大多数的文件读写，使用函数：

File\_put\_contents, file\_get\_contents完成。

少数，特别情况，以上两个函数不好用。

典型的就是，文件过大！只能一部分一部分的操作！

### Fopen()

打开文件句柄，PHP程序与文件间数据通路！

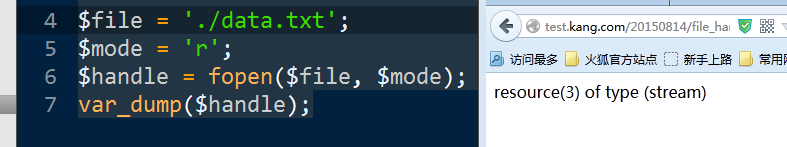
需要 文件地址 和 打开模式 两个参数！

打开模式：指定打开该文件后，需要执行什么操作！如下的模式可以选择：

R: read, 读模式

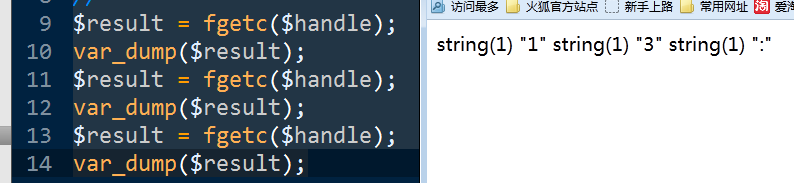
W: write, 写操作，清空写。

A: append，写操作，追加写。



### 读：Fread(),fgetc(),fgets(),feof();

#### 一个字节数据 = Fgetc(句柄); char， 字符！



1个字节：一个最基本存储空间

1个字符：a，康，X， 单字节字符（字母，标点，数字），存储需要消耗1个字节。多字节字符，存储需要消耗多个字节。占用的字节数量，与字符所采用的字符集相关！

字符集：（utf-8, gbk,ascii）

映射表：字符 与 存储编码 映射表

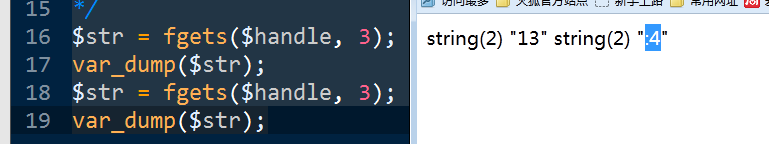
每当读取到一个字节，将文件指针向前移动一个字节！

句柄操作中，依赖的就是 文件指针！

#### 字符串 = Fgets(句柄, 长度)

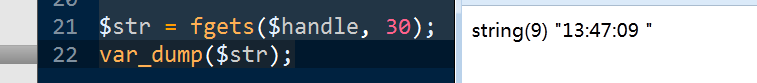
从文件指针位置，读取指定长度的字符串内容！

获取到的内容长度为 ，参数长度-1



同样会向前移动指针！

如果在读取时，先读到了换行符，也会终止。



也叫读行函数！

典型的逻辑，每行记录一组信息，该函数最常用！

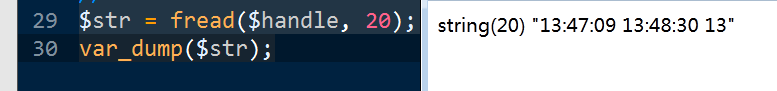
配合循环结构，读取全部记录。

需要： feof() END of File 用来判断是否到达文件末尾！



#### 字符串 = Fread(handle，长度)

依据长度读取内容，不受换行符的限制

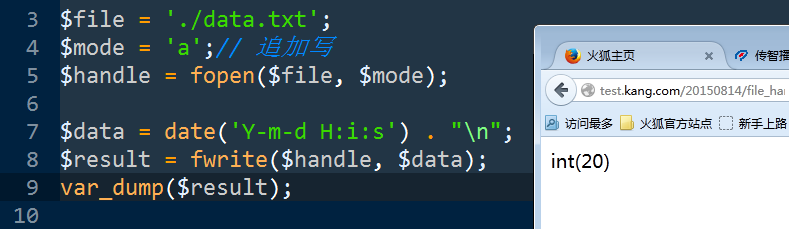


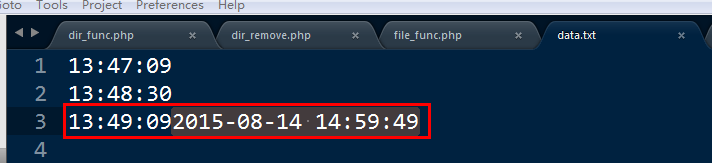
Tip：唯一一个限制为：最大的长度不能超过8192!

### 写：Fwrite();

写入长度 = Fwrite(句柄，内容)

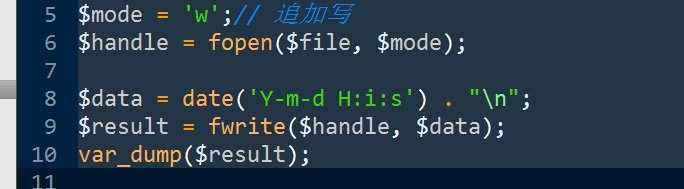
追加写：

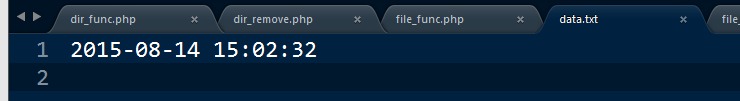




清空写：

是在建立句柄，打开文件时，就完成了清空的工作！



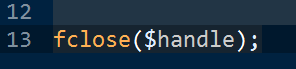


就是在文件指针位置进行写操作！

但是，如果为追加写模式，永远在末尾完成写操作！

### Fclose();

关闭句柄



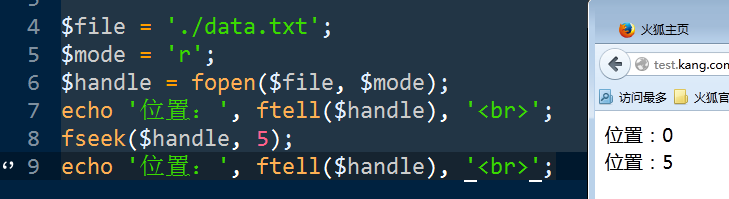
## 指针操作函数

### Fseek(句柄， 位置);

定位指针，位置从0开始，递增！

### Ftell(句柄);

获取指针位置！



## 打开模式

基本模式：

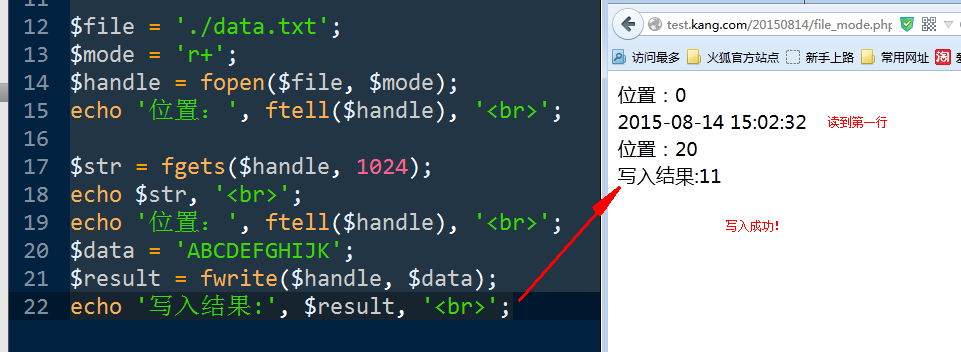
* R 读
* W 清空写，文件存在直接打开同时清空。
* A 追加写，文件存在直接打开
* X 新建写，只能新建文件进行操作！就是写操作，与w类似

+扩展模式：

扩展的操作，都可以完成读写操作。

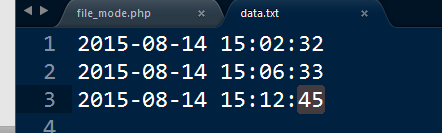
差异与基本模式一致！

* R+ 读写，打开任意文件（无论是存在还是不存在），文件内容不会被清空，依据指针位置，完成读，写操作。其中，写，会替换原有指针位置字节。

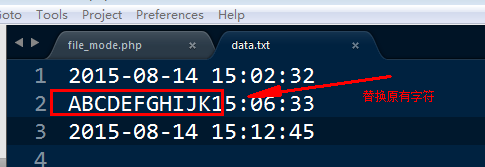


结果导致：

元数据：

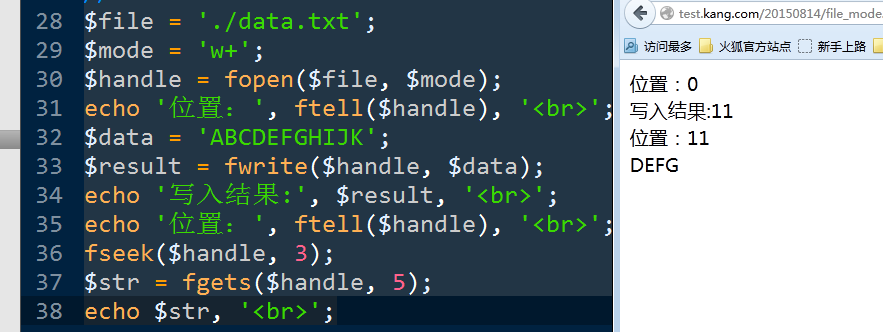


写入后：

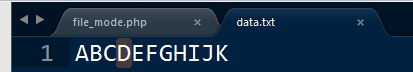


* W+ 读写，打开时，同时清空内容，之后指针在哪里，就 在哪里完成读写！与R+唯一的差异，就是会清空文件内容！

测试：

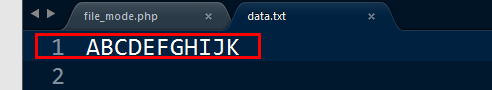


结果：

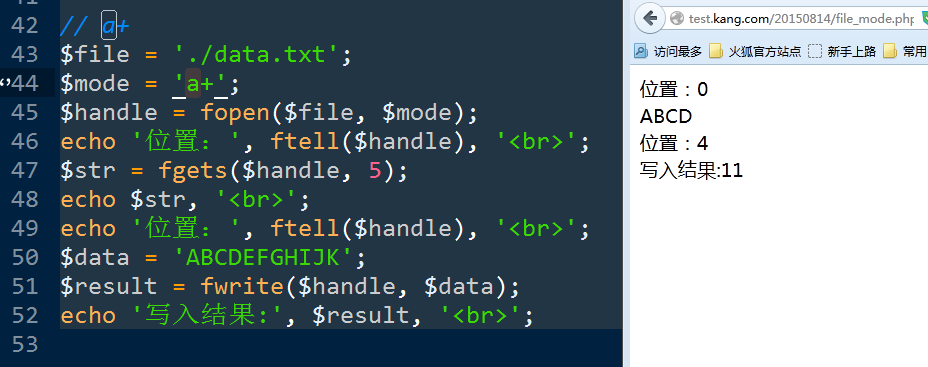


* A+ 读写，打开任意文件，不会清空内容，指针仅仅影响读操作。不影响写操作，仅仅可以在末尾写！

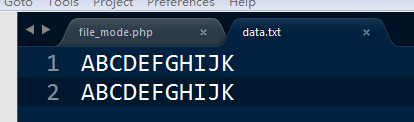
原始数据：



测试：



写操作在最后：



* X+ 读写，新建才能打开，依据文件指针位置，进行读写操作！

## 文件并发操作（文件锁）

默认的，PHP的文件操作函数，不是阻塞状态。自由操作状态。

当需要，一个脚本操作而需要阻塞另外的脚本操作时，需要用到文件锁。

锁操作流程：

先加锁，检测锁是否加成功，如果成功再使用！

锁定类型：

读锁： s-Lock(share-lock)共享锁，读操作前，期望增加的锁定。导致，允许并发读，阻塞额外的写操作。

写锁： x-lock（exclusive-lock）排他锁，独占锁，写操作前，尝试添加的锁定类型。导致，其他脚本不能读也不能写。

意向锁：所有的操作资源的脚本都遵循一个约定来使用文件锁（约定）。

函数：

Flock(句柄, 类型)

函数用于添加PHP 的文件锁定。（，添加意向锁。）

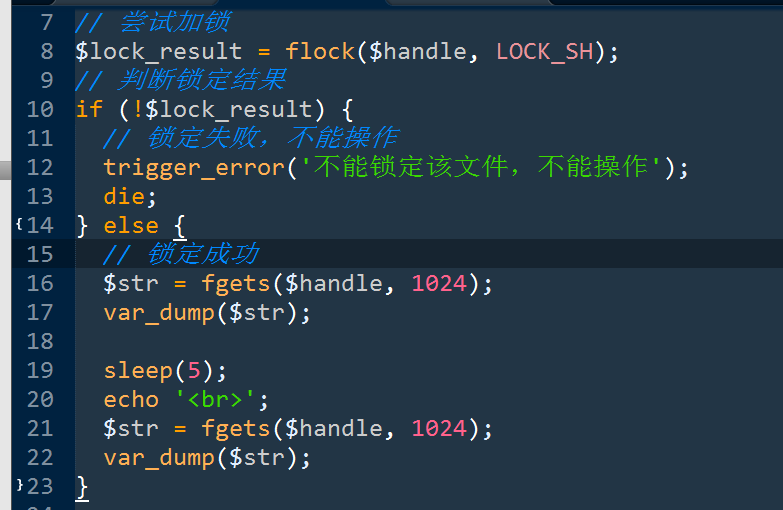
类型：

LOCK\_SH 读锁

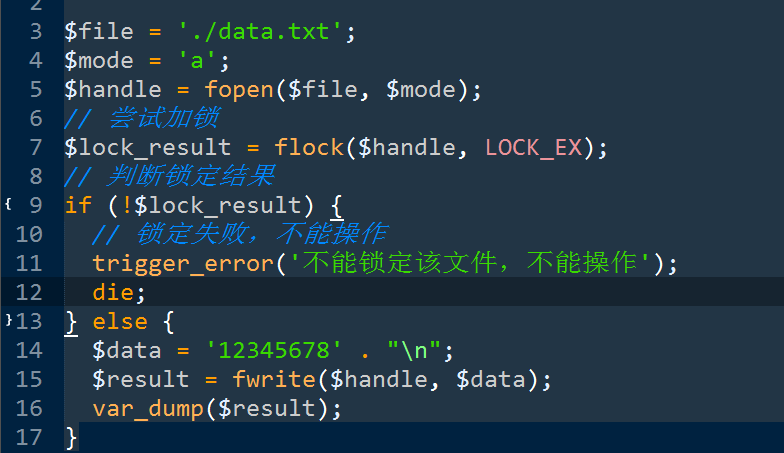
LOCK\_EX 写锁

语法如下：

1:



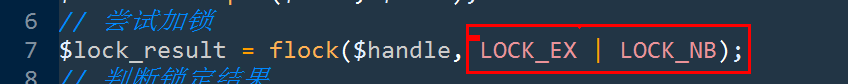
2:



2,需要等待（被阻塞）1 执行完才会执行！

强调，一定要判断锁是否可以添加成功！

可以通过LOCK\_NB来在锁定失败时，不阻塞：



解锁

Flock($handle, LOCK\_UN)来强制解锁，

Fclose()自动解锁！

# HTTP

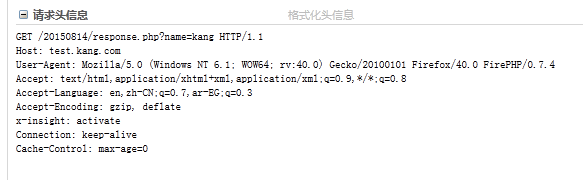
协议，约定，规范，一套语法！

HTTP: 超文件传输协议。应用层的协议。规范 浏览器与服务器 间数据交互格式！

规范：

请求数据格式，和 响应数据格式。

## 请求数据格式

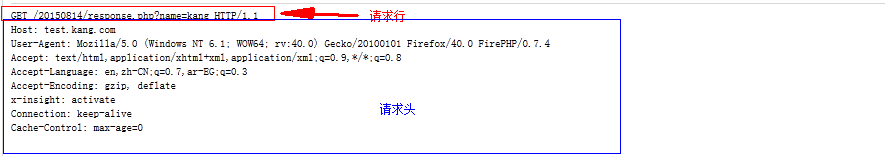


以上就是典型的get请求的数据，

浏览器（请求代理端）需要，将数据整理成以上的格式，向服务器端发送才可以！

以上数据共2个部分：

请求行request-line， 请求头request-header



还有上面的数据没有展示的部分：

请求主体request-content。

### 请求行

请求数据的第一行，请求的摘要信息。

表示：请求方法 请求的资源地址 使用的协议版本



### 请求头

浏览器向服务器传输的请求属性信息。浏览器需要服务器知道的浏览器的状态。

例如：

User-Agent: 用户代理，谁（什么）发出的这个请求。

Accept-Language：请求代理端（浏览器）可以接受的语言类型。

Accept：请求代理端（浏览器）可以接受的，内容类型

Accept-Encoding: 请求代理端（浏览器）可以接受的编码类型，指的压缩相关。

Host: 请求的主机名。

Connection：连接类型。http/1.1 新标准。响应结束是否立即断开TCP连接。

值：keep-alive， 短时间内保持连接。Close:立即关闭。

Cookie：携带的COOKIE！

语法上：

每行表示一个头。

行结束采用的是：回车+换行。 \r\n , 称之为CRLF

头由标识和内容组成，由冒号分隔！冒号要紧跟着标识。

头以空行结束。

### 请求主体

请求的主体数据。

通常在使用POST方式发出请求时，POST数据就在请求主体中传输！

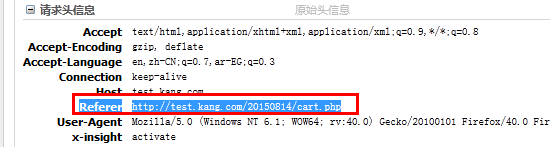
## 操作请求

在服务器端接受到浏览器请求后，利用相应的请求信息完成操作

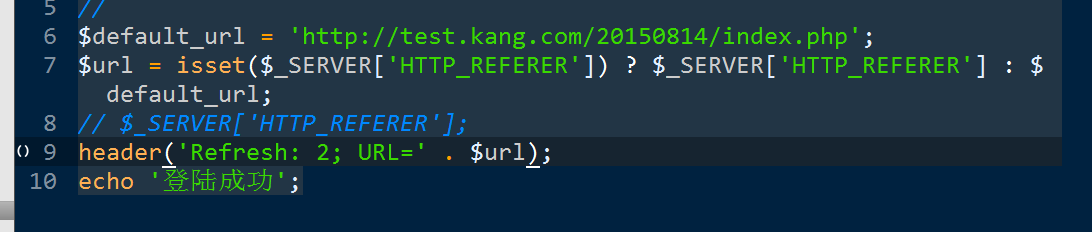
### 返回来源页

利用请求头当中的来源完成的：

Referer表示请求来源！



实现如下：



### I18n程序，国际化程序

多语言程序

i18n（其来源是英文单词 internationalization的首末字符i和n，18为中间的字符数）是“国际化”的简称

项目支持多种语言的展示。

为不同的浏览器需求展示不同的语言。

核心：如何知道浏览器需要哪种语言？浏览器请求时携带Accept-Language头，表示可以接受的语言！

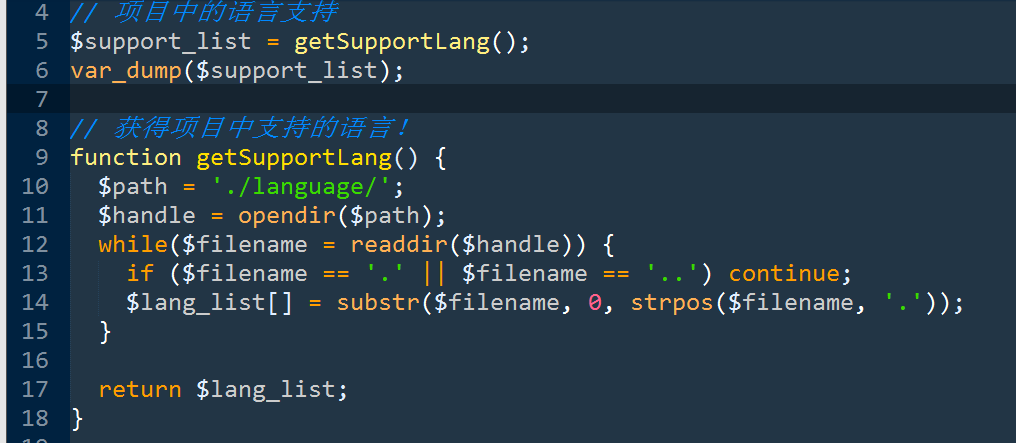
#### 项目支持多语言？

通过不同的翻译文件完成的。



Strpos(’abcda’, ‘a’) = 0

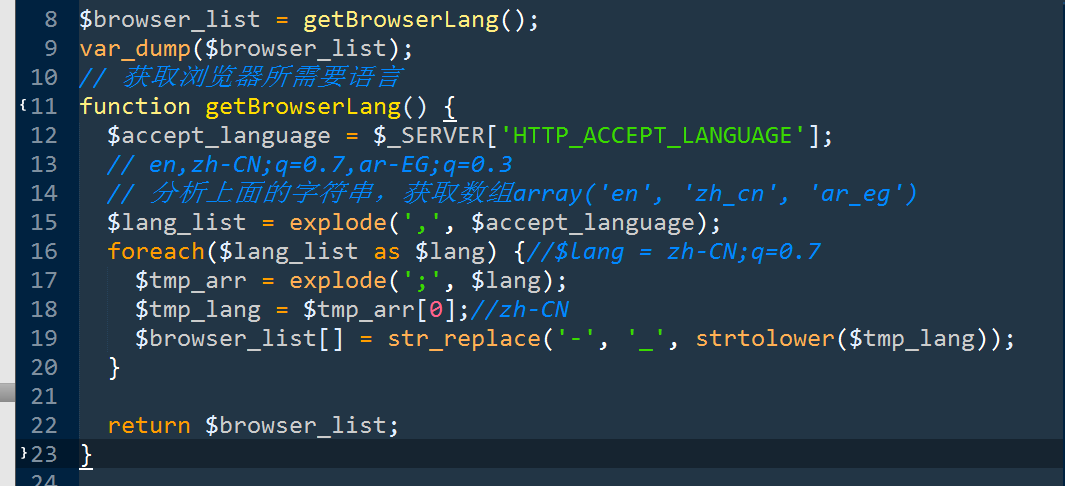
某个字符串首次出现的位置！



#### 浏览器需要哪种语言？

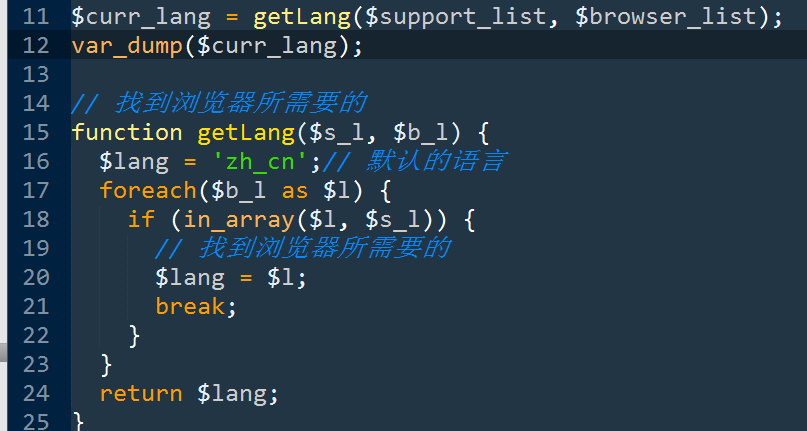
请求头信息中的Accept-Language

Str\_replace(‘查找’, ‘替换’, 字符串)

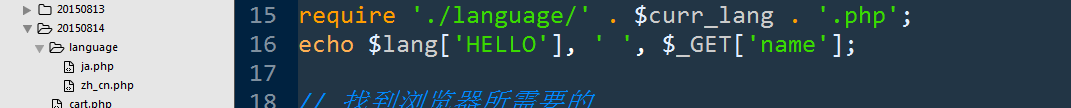


#### 确定响应语言

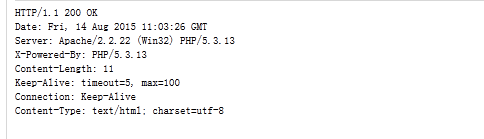
找到浏览器最希望的。



#### 利用语言文件，完成展示



## 响应的格式





典型的响应数据。

分成三部分：

响应行：响应的第一行

响应头：服务器需要浏览器知道一些信息。

响应主体：主体数据，用于展示！

### 响应行

协议版本 响应状态码 状态消息

状态码：响应结果。与状态消息一一对应的。

告知浏览器（请求代理端），当前响应的结果。

常用的状态码：

200 ok 成功

404 Not Found 请求资源不存在

403 Forbidden 请求被拒绝

302 Found 重定向

500 Server Internal Error 服务器内部错误，web服务器出了问题。

系列：

1XX，2XX，3XX，4XX，5XX：

1xx: 信息

2xx: 成功

3xx: 重定向

4xx: 客户端错误

5xx: 服务器错误

### 响应头

服务器需要浏览器知道的信息

Date：响应时间

Connection：连接类型

Keep-Alive：保持连接的实效

Content-Type：主体类型

Content-Length: 主体的长度（字节）

Set-Cookie：设置COOKIE的

语法与请求头一致：

CRLF行结尾

空行表示头结束

### 响应主体

任何 的输出都为响应主体。

Html代码，echo ’‘

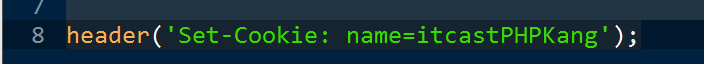
浏览器源代码中查看的内容，就是响应主体！

## 操作响应

Header()函数。

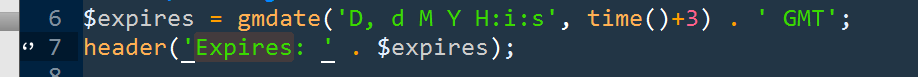
操作响应头的。

CookIe就可以：



任何 输出，html代码都是主体！

### 控制浏览器缓存



Expires控制响应的有效期的。

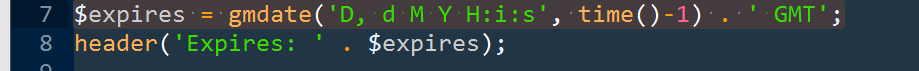
表示方式，特定格式的GMT时间：

|  |  |
| --- | --- |
|  | Fri, 14 Aug 2015 11:27:39 GMT |

Date()将一个时间戳，格式化成本地时间！

Gmdate()将一个时间戳，格式化成GMT时间。

如何告知浏览器，不要缓存。



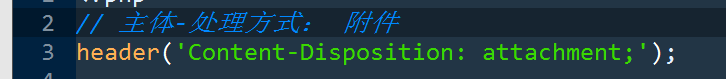
通常，验证码，程序，就需要告知浏览器，不要缓存验证码的图片！

### 文件下载

HTTP下载：

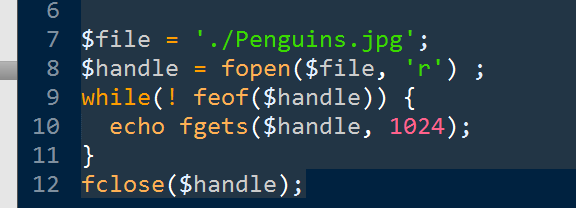
告知浏览器，将浏览器接受到的响应主体，以附件的形式进行存储！

通过响应头：

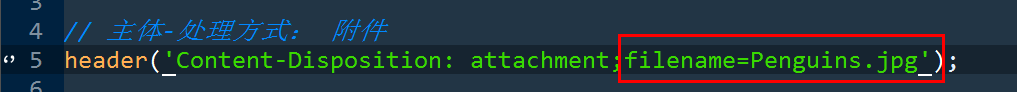


以下载图片文件为例：

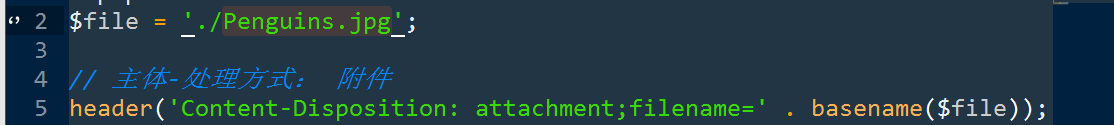
将图片内容，作为响应主体输出到浏览器端：



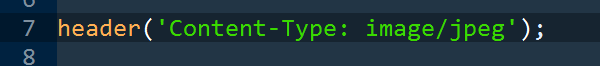
指定下载的文件名：

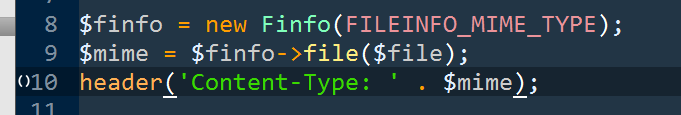


Basename()取得地址中的文件名部分。



文件类型：





文件大小：

