# 文件上传

上传：

从浏览器端传输的到服务器端。

请求时：

数据从浏览器端传输到服务器端。

可见：

上传，发生在浏览器向服务器发出请求过程中。

文件，对于浏览器来讲，就是表单中的一个特殊类型的数据而已。

浏览器表单中的数据：两种类型。

1. 字符串类型。（字节流编码）
2. 文件类型。（二进制编码）

（tip：文件就是表单中数据的一部分）。

提交表单时，浏览器自动将表单内的数据，传输到服务器端，文件型也会包含在内。

服务器角度：

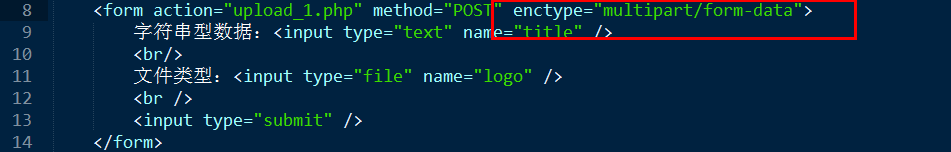
在接受浏览器的请求时，处理好表单内数据。

根据数据类型不同（2种），采用不同的处理方法：

1. 字符串类型，存储在$\_POST变量中（内存中）
2. 文件型数据，存储到上传临时目录中。

表单提交时：浏览器会默认的认为，表单内的数据都是字符串类型（即使使用file文件域），通过在form上，增加属性，告知浏览器表单内的数据类型不仅仅有字符串。

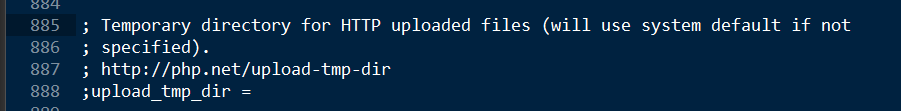
Form enctype=”multipart/form-data”



PHP服务器在接受到文件型的表单数据后：

将文件，存储于上传临时目录。就是临时文件，有效期脚本周期内有效。

通过PHP的php.ini配置：



默认为服务器操作系统临时目录。

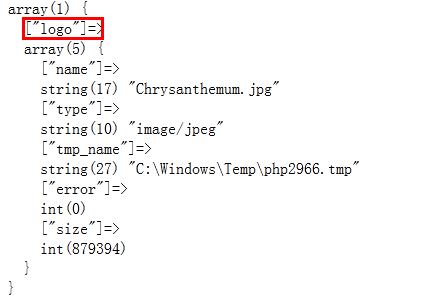
需要，将临时上传文件持久化存储！（将其移动到指定位置）

函数：

Move\_uploaded\_file(上传临时地址，目标地址);

完成

需要使用$\_FILES数组，存储了上传的临时文件的信息，包括临时地址！



每个上传的临时文件，有5个信息，name：原始文件名，type：类型，tmp\_name，临时地址，error：是否有错误及错误类型，size：大小。

## 典型的上传代码

通过对文件的属性信息的判断，决定文件是否可用！

类型：表示方式 2种：

后缀名：文件系统中，标识文件的方案。

MIME（多用途internet邮件扩展）：网络上传输资源时，使用的资源类型表示方法。

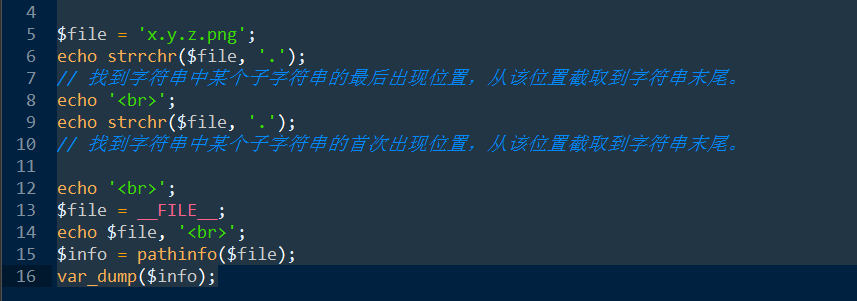
Text/html; image/png

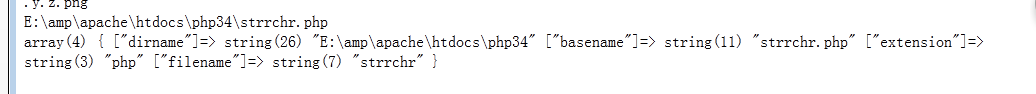
后缀名截取：

$file = 'x.y.z.png';

echo strrchr($file, '.');

// 找到字符串中某个子字符串的最后出现位置，从该位置截取到字符串末尾。





Array\_merge()数组合并

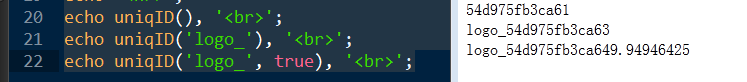
array\_unique()去掉数组中重复的元素值

重命名文件：

使用原名。（重名，特殊字符， 不具备逻辑含义）

起名：uniqID();

UniqID()可以生成一个唯一的字符串，并且允许设置前缀。



类型检测：

后缀和MIME

以上两个值，都是由浏览器端请求提供的。

需要服务器PHP自身完成MIME的检测：

需要 PHP的扩展： fileinfo 完成对文件的信息的检测。

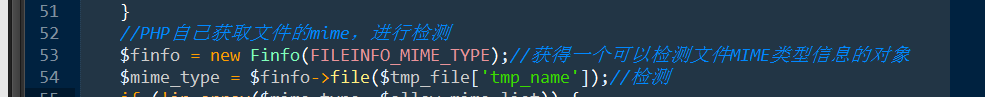
提供面向对象和函数两套语法：

使用面向对象语法吧：

开启该扩展：



Apache restart



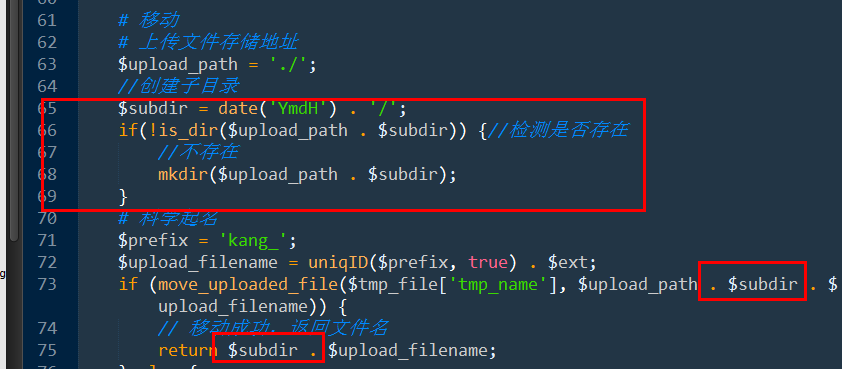
分子目录存储上传文件

原则：业务逻辑。文件数量。按照时间。

Mkdir(目录地址)创建目录

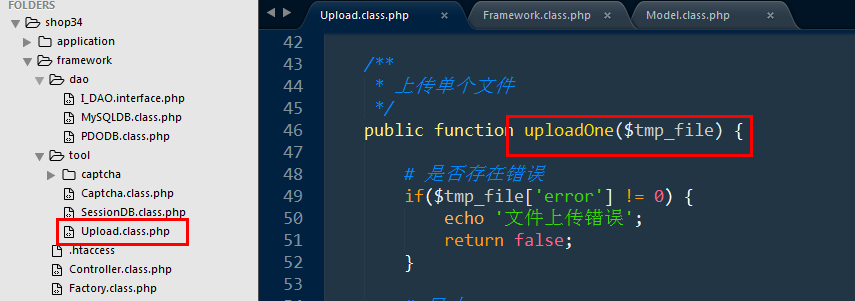
Is\_dir(目录地址)检测目录

Date()获得日期时间



## 项目中使用

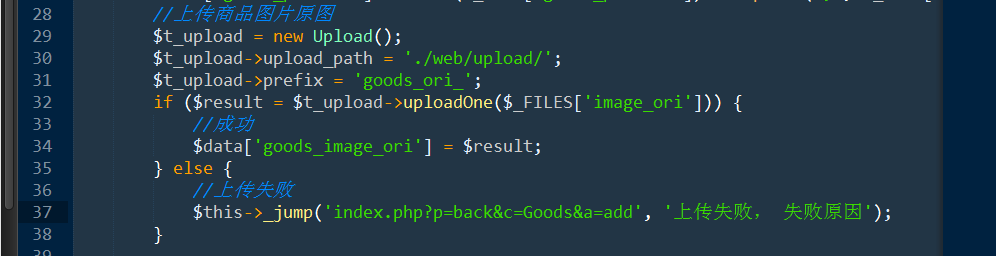
上传的工具类：



完成自动加载：

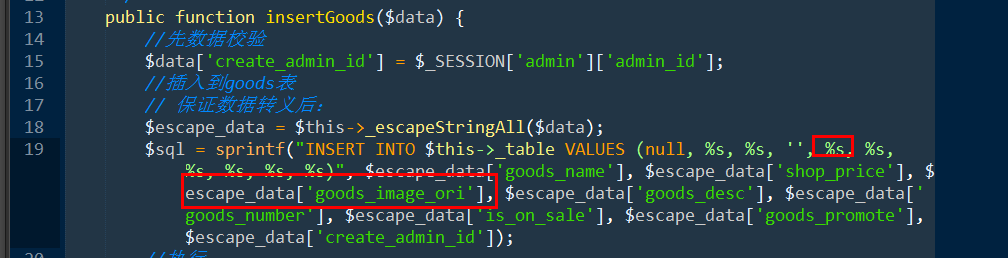
添加商品时使用：

Back/GoodsController->insertAction();



模型中，处理goods\_image\_ori:

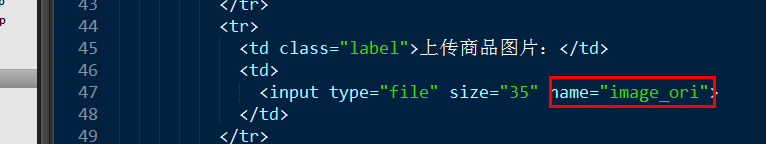
Back/GoodsModel->insertGoods();



修改表单，使之可以上传文件，并设置file表单元素，起名字。

Back/iew/goods\_add.html



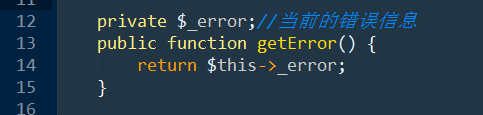


## 错误处理

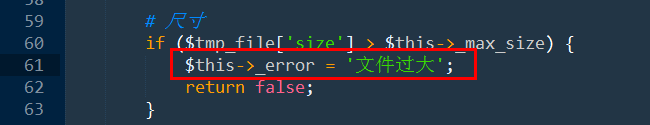
工具类，检测到错误，将错误信息记录。

需要报错，才通过控制器动作的跳转提示，将错误信息输出

工具类：



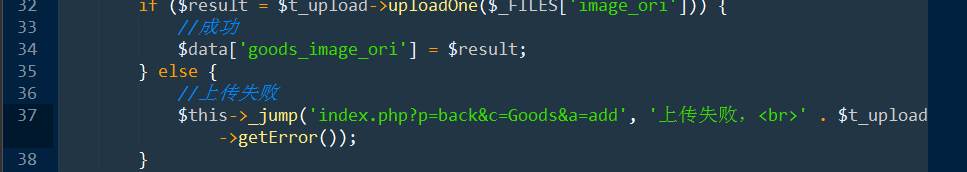
记录消息：



通过方法的返回值，判断执行结果。如果失败通过工具对象的getError()方法，获取错误信息！

在控制器中：

Back/GoodsController->insertAction();



Tip:

所有的工具类，都可以采用类似的方案处理错误消息！

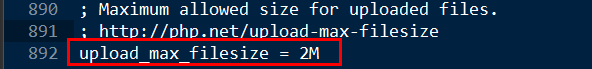
## 错误类型

Error元素表示错误，及类型

0123467可能值

0 表示无错误

1：文件过大，大于php的配置



2：文件过大，超过了表单元素max\_file\_size



Tip:目前也是服务器PHP判断的。

但是：

PHP提出的，希望，当文件过大时，在浏览器请求时就可以利用该值进行判断大小

3：上传部分文件。

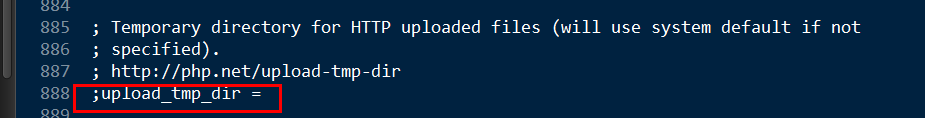
4：没有上传文件。

5：逻辑上，上传的为空文件（长度为0）

6：没有找到临时上传目录（权限控制）。

7：临时文件写入失败（磁盘空间，权限）。

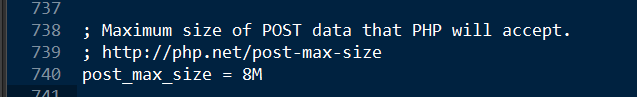
配置上传临时目录：



一次请求可以上传的文件数量：



POST数据存在最大值限制：

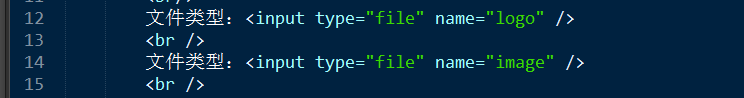


## 多文件上传

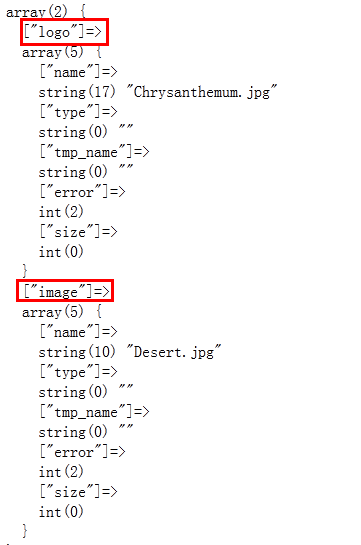
一次请求，存在多个文件。

表单中的文件域的name属性命名方式，导致$\_FILES的结构不同：

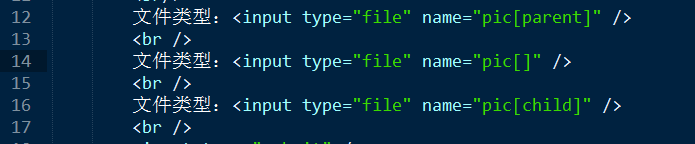
Name属性值独立不同：表示逻辑上 多个文件没有联系！



每个上传文件生成一个$\_FILES的元素，分别处理即可：

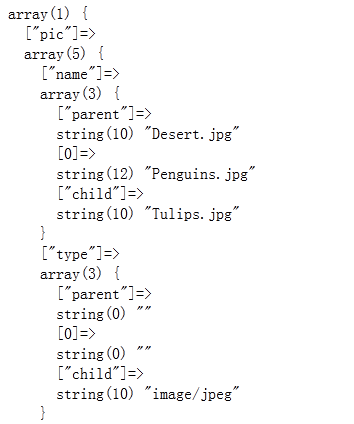


Name使用的数组方式命名：逻辑上传文件存在关联



提交后的$\_FILES的结构：

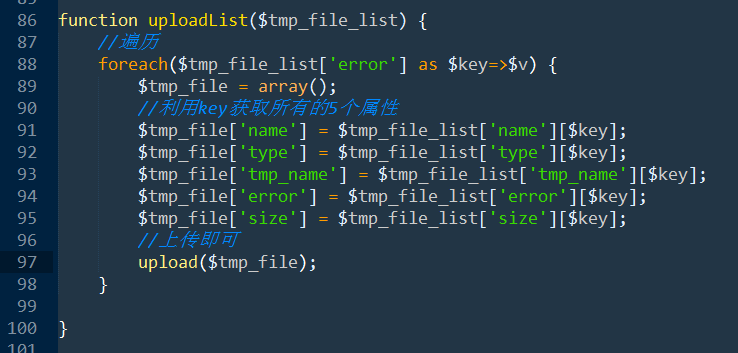
将所有一个组内的文件信息，整理到一个$\_FILES元素内，该元素也有5个（name，type，tmp\_name，size，error）元素组成。每个元素内，是数组，存储所有的名字，类型，等：



没有办法直接获取某个文件的上传信息：

需要，想办法拼凑出来每一个上传文件的信息，逐一处理：

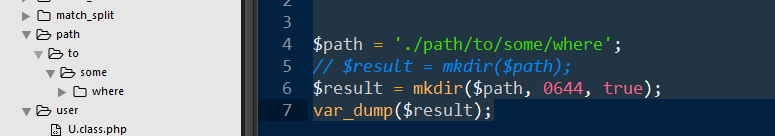
遍历任何一个属性(name,error),获取当前文件的key（标志），利用该相同的key，从其他的属性中获取当前文件的其他信息（共5个即可）



# 目录操作

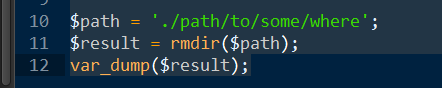
## 创建目录

Mkdir(目录地址，权限，是否递归创建=false);



## 删除目录

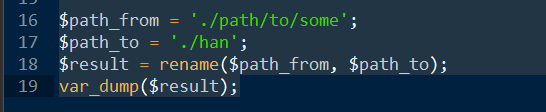
Rmdir(目录地址)



仅仅可以删除空目录。（不支持递归删除）

## 移动（改名）

Rename(旧地址，新地址)

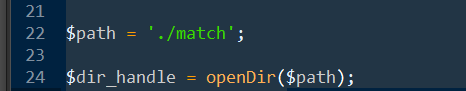


该函数也支持操作文件。

## 获取目录内容（文件）

打开目录 目录句柄 = opendir(目录地址)

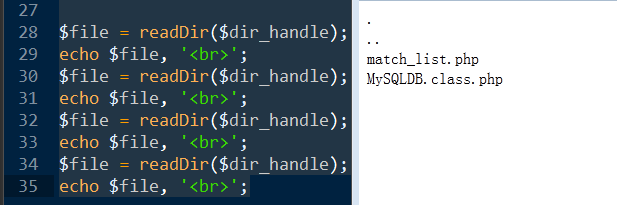
目录句柄：php程序与目录间的一个联系。



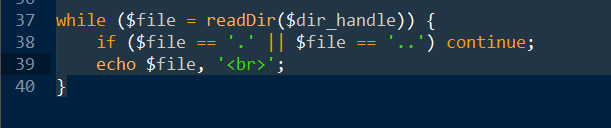
依次读取目录内文件 文件名 = readdir(目录句柄);

依次读取一个文件（子目录），同时向下移动句柄指针。如果读取不到返回false！

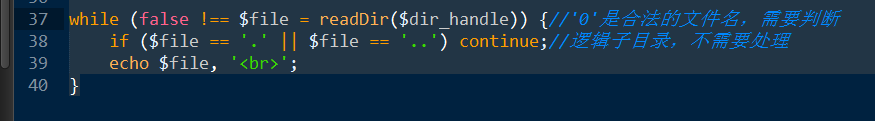
目录内存在两个逻辑子目录 ., .. 分别表示当前目录和上级目录！



通过配合 循环结构 完成目录内所有文件的读取：



处理 0 文件名：



关闭目录 closedir()



## 递归获取目录内容

打开某个目录

依次读取目录内文件

如果某个文件为目录

递归对该目录采用打开，读取，判断这个递归草

递归点： 如果子文件为目录，则递归

出口： 如果目录中不存在子目录，则不需要继续递归！



### 树状展示

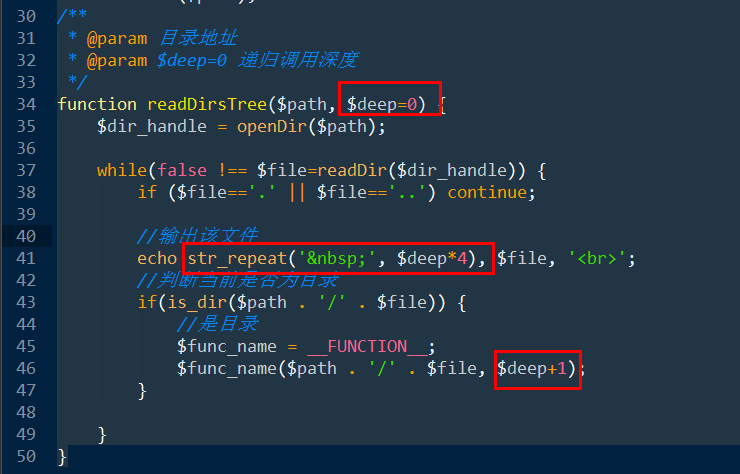
使用特定数量的缩进达到树状目的！

核心问题，计算需要缩进的数量！

缩进级别，与递归调用深度保持一致。每当执行一级递归操作，所找到的文件的缩进级别+1;

语法实现：

增加一个参数，表示当前函数调用的深度级别，每当递归+1;



### 嵌套

使用多维数组的嵌套，表示层级关系！

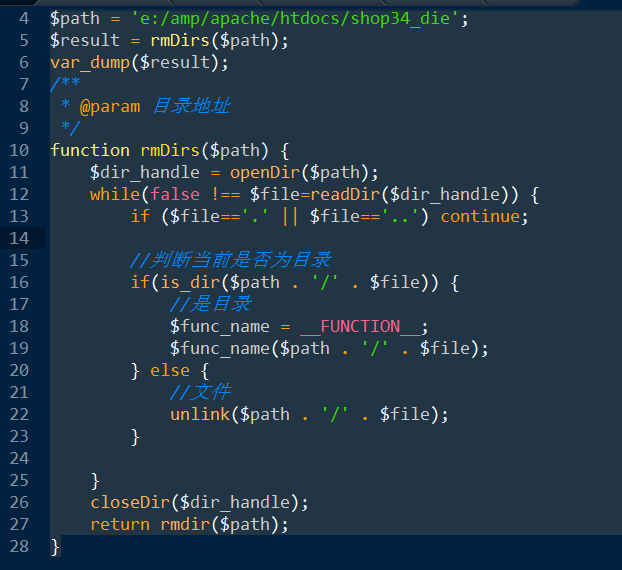
例如：



实现：

## 递归删除

Unlink(文件地址)删除文件。



目录文件名编码问题：

展示时，将操作系统编码转换为响应数据编码。

Windows gbk，项目utf-8



代码的地址中存在中文：需要转换为系统编码，才能被文件系统识别：

