12-1

Invalid 无效 calculate 计算 expire 到期 auto自动的 generate 生成 convenient 方便的

学习文件上传原生

HTML调用JS JS调用PHP

MVC 框架基础

12-4

新版的source tree 有ssl的bug

开了一个cpmad项目 仿制madsck

12-5

Trait 特性

（3-13添加）,trait不算是一个类，在这个当中写的方法，假如想要使用，是要在你要使用的类中use一下这个trait的名字，就能够使用了。而顺序是这个类本身的方法可以覆盖trait里面的方法，而trait的方法又可以覆盖掉当前类继承的类的方法。

// 路由

Route::get('user/profile', 'UserController@profile')->name('profile');

// 產生對應網址

$url = route('profile');

// 轉址

return redirect()->route('profile');

// 根據 name 轉址

redirect()->route('profile');

// 根據網址

redirect()->to('user/profile')

12-6

分页用数据库的groupBy 还有PHP的array\_slice

或者直接覆盖掉pagination

样式可以在F12的source里面直接复制

12-7

熟悉了CSS的一部分，对float有了了解。

准备学习一点JS基础

12-8

知道了基本的JS语句实现

12-9

路由传参用正则约束，不用正则约束那就在控制器里验证，验证不通过抛出一个异常，或者来到404页面。

12-12

GetOriginal获取原始的值

GetAttribute 获取现在的值 可以在model当中进行预处理

这两个可以通过dd看到

处理图片带上with（3-13补充可以不用带上with写法）

行和列 row和column

+代表public，-代表private，#代表protected，属性后面灰色的部分代表默认值。

12-14

http\_build\_query 把原有的搜索条件加上去，可以使用这个函数

input里面的type=“hidden” 是用来隐藏输入的，很有用！！！！！！！

12-15

Option的selected属性的时候可以在option当中进行判断，比如说@if（isset(XXX)）selected @endif

因为selected会把当前适用的选项预加载，所以有匹配到他就直接先加载这个selected

Self可以访问本类的静态方法和属性，以及访问父类的静态方法和属性，用self的时候可以不用实例化。

12-19

1. **Model** :仅当成Eloquent class。
2. **Repository** :辅助model，处理资料库逻辑，然后注入到service。
3. **Service** :辅助controller，处理商业逻辑，然后注入到controller。
4. **Controller** :接收HTTP request，调用其他service。
5. **Presenter** :处理显示逻辑，然后注入到view。
6. **View** :使用blade将资料binding到HTML。

Count on 指望

12-20

在store当中使用Validator make一个实例，然后在下面进行判断随后抛出一个异常，这个异常也就是$errors这个变量，这个变量是一个messagebag实例，而这个变量能够被所有视图自动的获取，所以只需要增加增加一个报错的信息就行。比如增加一个error.blade.php 里面对这个$errors进行一些处理，显示出来。

12-21

匿名函数：anonymous functions，也叫闭包函数closures，允许临时创建一个没有指定名称的函数。经常用作回调函数

<?php  
$greet = function($name)  
{  
    printf("Hello %s\r\n", $name);  
};  
  
$greet('World');  
$greet('PHP');  
?>

将一个没有名字的匿名函数赋值给一个变量，而这个变量也可以当做一个方法来使用

*function* callback($callback)  
{  
 $callback();  
}  
  
callback(*function*() {  
 *print* "This is a anonymous function";  
});

事先定义了一个callback函数，里面有一个变量callback，而这个变量就是下面callback当中的一个方法，使用函数的使用方法就能够直接调用。

例子

<?php

/\*\*

\* author: selfimpr

\* mail: lgg860911@yahoo.com.cn

\* blog: http://blog.csdn.net/lgg201

\* 下面提到的代码在PHP5.3以上版本运行通过.

\*/

function callback($callback) {

$callback();

}

//输出: This is a anonymous function.<br />/n

//这里是直接定义一个匿名函数进行传递, 在以往的版本中, 这是不可用的.

//现在, 这种语法非常舒服, 和javascript语法基本一致, 之所以说基本呢, 需要继续向下看

//结论: 一个舒服的语法必然会受欢迎的.

callback(function() {

print "This is a anonymous function.<br />/n";

});

//输出: This is a closure use string value, msg is: Hello, everyone.<br />/n

//这里首先定义了一个闭包, 这次户口本上有名字了...

//use, 一个新鲜的家伙...

//众所周知, 闭包: 内部函数使用了外部函数中定义的变量.

//在PHP新开放的闭包语法中, 我们就是用use来使用闭包外部定义的变量的.

//这里我们使用了外部变量$msg, 定义完之后, 又对其值进行了改变, 闭包被执行后输出的是原始值

//结论: 以传值方式传递的基础类型参数, 闭包use的值在闭包创建是就确定了.

$msg = "Hello, everyone";

$callback = function () use ($msg) {

print "This is a closure use string value, msg is: $msg. <br />/n";

};

$msg = "Hello, everybody";

callback($callback);

//输出: This is a closure use string value lazy bind, msg is: Hello, everybody.<br />/n

//换一种引用方式, 我们使用引用的方式来use

//可以发现这次输出是闭包定义后的值...

//这个其实不难理解, 我们以引用方式use, 那闭包use的是$msg这个变量的地址

//当后面对$msg这个地址上的值进行了改变之后, 闭包内再输出这个地址的值时, 自然改变了.

$msg = "Hello, everyone";

$callback = function () use (&$msg) {

print "This is a closure use string value lazy bind, msg is: $msg. <br />/n";

};

$msg = "Hello, everybody";

callback($callback);

//输出: This is a closure use object, msg is: Hello, everyone.<br />/n

//闭包中输出的是之前被拷贝的值为Hello, everyone的对象, 后面是对$obj这个名字的一个重新赋值.

//可以这样考虑

//1. obj是对象Hello, everyone的名字

//2. 对象Hello, everyone被闭包use, 闭包产生了一个对Hello, everyone对象的引用

//3. obj被修改为Hello, everybody这个对象的名字

//4. 注意, 是名字obj代表的实体变了, 而不是Hello, everyone对象, 那自然闭包的输出还是前面的Hello, everyone

$obj = (object) "Hello, everyone";

$callback = function () use ($obj) {

print "This is a closure use object, msg is: {$obj->scalar}. <br />/n";

};

$obj = (object) "Hello, everybody";

callback($callback);

//输出: This is a closure use object, msg is: Hello, everybody.<br />/n

//还是按照上面的步骤, 按部就班的来吧:

//1. obj名字指向Hello, everyone对象

//2. 闭包产生一个引用指向Hello, everyone对象

//3. 修改obj名字指向的对象(即Hello, everyone对象)的scalar值

//4. 执行闭包, 输出的自然是Hello, everybody, 因为其实只有一个真正的对象

$obj = (object) "Hello, everyone";

$callback = function () use ($obj) {

print "This is a closure use object, msg is: {$obj->scalar}. <br />/n";

};

$obj->scalar = "Hello, everybody";

callback($callback);

//输出: This is a closure use object lazy bind, msg is: Hello, everybody.<br />/n

//闭包引用的是什么呢? &$obj, 闭包产生的引用指向$obj这个名字所指向的地址.

//因此, 无论obj怎么变化, 都是逃不脱的....

//所以, 输出的就是改变后的值

$obj = (object) "Hello, everyone";

$callback = function () use (&$obj) {

print "This is a closure use object lazy bind, msg is: {$obj->scalar}. <br />/n";

};

$obj = (object) "Hello, everybody";

callback($callback);

/\*\*

\* 一个利用闭包的计数器产生器

\* 这里其实借鉴的是python中介绍闭包时的例子...

\* 我们可以这样考虑:

\* 1. counter函数每次调用, 创建一个局部变量$counter, 初始化为1.

\* 2. 然后创建一个闭包, 闭包产生了对局部变量$counter的引用.

\* 3. 函数counter返回创建的闭包, 并销毁局部变量, 但此时有闭包对$counter的引用,

\* 它并不会被回收, 因此, 我们可以这样理解, 被函数counter返回的闭包, 携带了一个游离态的

\* 变量.

\* 4. 由于每次调用counter都会创建独立的$counter和闭包, 因此返回的闭包相互之间是独立的.

\* 5. 执行被返回的闭包, 对其携带的游离态变量自增并返回, 得到的就是一个计数器.

\* 结论: 此函数可以用来生成相互独立的计数器.

\*/

function counter() {

$counter = 1;

return function() use(&$counter) {return $counter ++;};

}

$counter1 = counter();

$counter2 = counter();

echo "counter1: " . $counter1() . "<br />/n";

echo "counter1: " . $counter1() . "<br />/n";

echo "counter1: " . $counter1() . "<br />/n";

echo "counter1: " . $counter1() . "<br />/n";

echo "counter2: " . $counter2() . "<br />/n";

echo "counter2: " . $counter2() . "<br />/n";

echo "counter2: " . $counter2() . "<br />/n";

echo "counter2: " . $counter2() . "<br />/n";

?>

12-25  
搜索框string字段验证假如想要这个字段为空请在验证当中加上nullable

FAQ frequently asked question 常见问题

如何问问题：<https://github.com/ruby-china/How-To-Ask-Questions-The-Smart-Way/blob/master/README-zh_CN.md>

12-26

HTML已经加载之后再使用JS，那么他就会覆盖整个文档。

大小写分明，JS分号结尾可选，根据在HTML中的编写顺序执行，JS当中更多的是camelCase 而不是camel\_case。

将值赋给未声明变量，这个值将会自动变为全部变量

Element 元素

12-27

Syntax 语句

创建对象时没有初始值，那么都会是false。

MDN

12-28

Previous 以前

关于JS 的放置位置 可以参考的文章有：<https://www.zhihu.com/question/34147508>

网页是从上到下进行加载的，所以把JS放在网页抬头会导致页面加载时间过慢。尽量把JS放在网页的尾部

使用console.log()进行调试

12-29

JS 字符串操作 数组操作