## 框架使用指南

前言:

此框架使用的 数组是[ ]的形式,还采用了命名空间等特性,所以安装的PHP版本必须大于5.5以上, 不然是不可以使用的哦 \_—\_ 。

使用方法

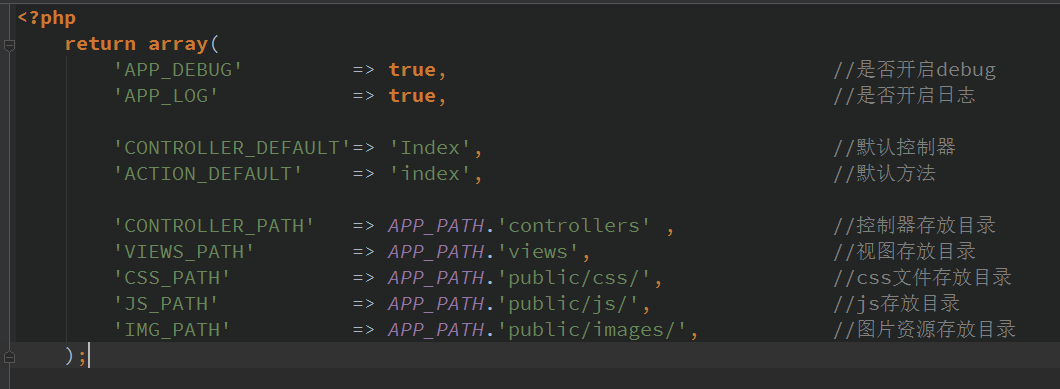
直接下载此框架放到你的坏境目录下,也可以放到配置好虚拟主机上(配置好的主机首页一定要是index.php) 地址直接指向根目录即可。访问方式是 你的服务器地址 例如youservername 在浏览器输入youservername 则会进入到欢迎页面 welcome 则是配置成功了, 默认访问的是 index控制器下的index方法 例如：youservername/index/index/key/value 红色的是配置的域名、黑色是控制器名、蓝色是方法名 key是GET键名 value是 key的值 index控制器下的index方法携带了 key=value的一个参数可以使用get()方法接收这一个参数。

说明:

url美化 隐藏了index.php 不过你的apache要开启重写模式 nginx也是需要配置 不知道的可以百度一下 比较简单 在这里不做过多的介绍

安全方面：

在配置文件 config/app.php里定义了一个数组 把值改成false之后,看不到任何的错误信息,建议在开发阶段改成true 项目上线部署的时候改成false;



日志:

Config/app.php数组里的APP\_LOG改成true会写入日志、false不会进行写入日志。注意点 在liunx写入日志时需要权限,遇到此类问题 修改日志文件的权限即可 chmod -R 777 log/

控制器:

控制器一般存放到 app\controllers目录下 建立控制器 请参考Index控制器的规范 文件名必须与类名相同首字母大，命名空间 namespace app\controllers; 如果有需要使用父类控制器的方法,则需要继承core\lib\controller父类即可 方法命名随意(不区分大小写)如index list update 访问时 youservername/控制器名/方法名/键1/值1

父类控制器的方法简介 可以通过$this调用

view() 方法 载入模版文件 并且可以传递参数 有两个参数

1、是模版文件名(后缀名可带可以不带) 但是模版文件一定是以.php为后缀

2、具体的参数 必须以数组的方式传递

例如 $arr = ['id'=>1]; $this->view('index',['arr'=>$arr]);

在模版输出的时候需要注意 示例 假如有一个数组 $arr= ['id'=>1]; 输出时 直接输出$arr['key']即可 echo $arr['id']; 即可得到 1这个值

session() session的增删改查

@param string $key session名

@param bool $value session值

@param bool $time 生存时间

@return int|string

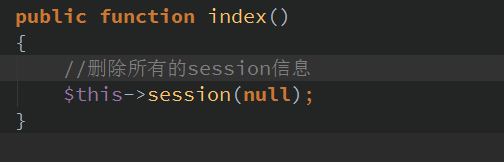
详解：参数1为字符串 当不为空且只有这一个参数时 返回当前名为参数1的session值不存在返回false;

参数2为session值例如$this->session('key','value') 设置了一个名为 key的 值为 value的一个session

参数3为session的生存时间 默认不设置session的过期时间

例如:$this->session('key','value',12); 设置了一个12秒过期的session信息

当调用session方法时 如果第一个参数为null 则是清除所有的session信息



删除单个session信息 第二个参数设置为null即可



cookie() 的方法与session的方法类似 可以参考 上面关于session方法的介绍

@param string $key cookie名

@param string $value cookie值

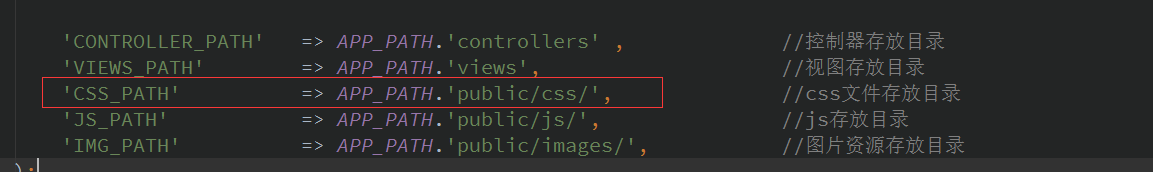
@param string $time 生存时间

@return string

静态资源

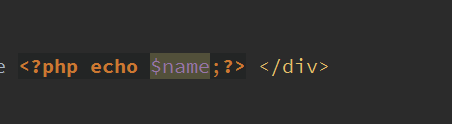
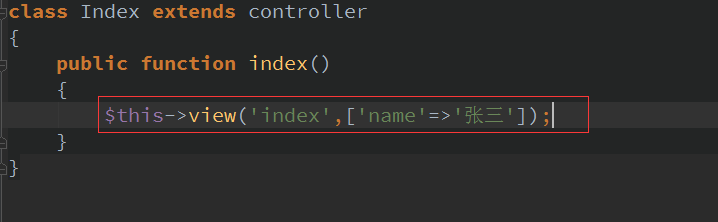
默认放在app/public下面,在配置文件目录下config/app.php可以自定义存放目录,目录存放的位置一个数组键和值

可以把数组的键名,当作常量来输出,例如输出css和js和图片的所在的目录 可以这样输出 <?php echo CSS\_PATH ?>



视图文件

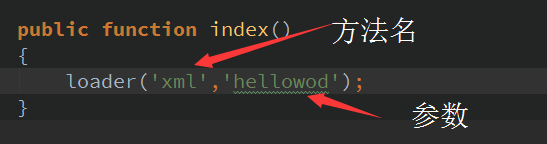
文件目录在app/views/下面 具体的不做要求 也可以新建目录 控制器加载模版时 请拼上 文件夹名/.../模版名 后缀名为php 可加可不加 例如 test/index.php 也可写成 tset/index 注意点(模版后缀统一为php) 如果控制加载模版时 参数为模版名时 会直接加载views下面的模版 例如 ('index') 会直接加载views下面 index.php 模版 如果为('test/index') 会加载 app/views/test/index.php的模版 控制器调用模版请使用控制器父类的view()方法 $this->view(‘index’); 有需要传递参数到模版中 请view()方法的第二个参数 $this->view(‘index’,[‘name’=>’zhangsan’]); 视图模版中输出方法是 <?php echo $name ?> 示例如下



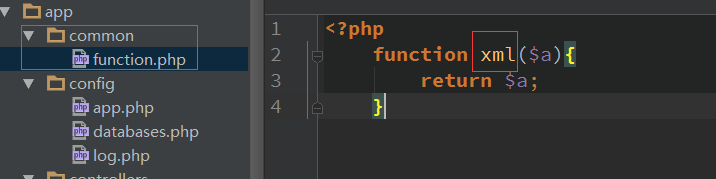
注意点：参数必须要以数组的形式进行传递

自定义函数:

可以写到app/common/function.php里面即可 用的时候请使用loader(方法名,参数)方法 参数为函数名 例如loader('funcname',$arr)参数1为方法名 参数2为具体的参数可以为array|object|string. 示例:



实际调用的是 common/function.php 里的 xml()方法。



错误模版 error:

在配置文件app.php中,把APP\_DEBUG改成false,出现异常会加载 app/views/error 里面的一些错误模版,可以进行修改,也可以不修改

配置文件:

配置文件放在了 app/config文件中 一般的配置信息都可以在里面根据实际情况下进行修改, 例如:数据库配置信息 路由配置信息 log日志配置信息 项目配置信息等

全局函数详情

全局函数放在了 core/common目录下 名为function.php

下面进行关于所有函数的详解及使用

p() 函数 用来打印出 string|array|bool 参数只有一个 那就是要输出变量

post() 函数 用来接收post数据的 参数有三个

a、 第一个参数 是具体的POST数组中的key 例如 post('name') 等于 $\_POST['name'] 返回的是具体的值

b、 第二个参数 默认值 如果调用 post这个函数 没有找到其中的key会返回第二个参数的值 也就是自定义的默认值 可以不设置默认返回false

c、 第三个参数 验证规则 string或者int 默认不进行验证(注意 但也会进行相关安全过滤等 请放心) 验证不通过时会返回 第二个参数

post函数 也可以不传递任何参数 返回的是一个数组 也是用户提交post的所有数据 (安全方面 也进行了相关的过滤 例如 特殊字符转义 删除html标签或者删除js代码等等...)

get() 与post函数类似 是接收get数据的 也有三个参数

a、第一个参数是 $\_GET参数的 key 可以为空 为空则返回所有的get的参数 没有参数则返回false;

b、第二个参数是返回自定义的默认值 不传默认为false;

c、第三个是验证规则 int|string 数字或者字符串 默认不进行验证 验证通过返回 设置的默认值

安全方面 如果值里面包含特殊字符 会被返回默认值 也就是第二个参数 get参数被容许传递的值只有大小写字母和数字

redirect() 参数有三个 网页的跳转

1、第一个参数是 跳转的地址 默认为false 如果不填则返回上一页 填了则跳转到相应的地址

2、第二个参数是 等待的时间 默认为三秒 3秒后跳到相应的地址

3、第三个参数是 显示的提示信息(会在等待的时间内看到这个提示信息) 但是如果设置第二个参数为0或者为false 是看不到的提示信息的 因为是进行了直接的跳转了

msubstr() 截取中文字符 有四个参数 1个参数是必填的 那就是第一个参数 要截取的字符串

1、第一个参数是 要截取的字符串 可以为中文也可以为字母和数字

2、第二个参数是 截取的开始位置 默认为0意思是从第一个开始截取

3、第三个参数是 截取的终点位置 默认是截取到最后

4、第四个参数是 字符的类型 默认为 utf-8 也可以设置为gbk|gb2312|big5

返回的是 截取完成后的字符串

removeJsStr() 删除js代码 只有一个参数 就是要被删除的js代码的字符串

返回的是删除后的 字符串

fileUpload() 附件上传

@param1 string $path 上传的路径 必须要的参数

@param2 array|string $file 文件资源 可不传会自动获取到$\_FILES里的参数

@param3 int $max 如许上传的文件最大限度 单位 m 默认2m

@param4 string $fileName 文件名 可以设置上传成功后的附件名称 不传有默认设置

@param5 array $type 可以上传的类型 有默认设置 也可以设置

@return bool|string 失败或者成功的路径

xmlToArray() xml数据转成普通数组

参数1是一个xml数据

loader() 调用自定义函数

参数1 是一个函数名的字符串

参数2 是调用这个函数参数 可不传这个参数 loader('xmlToArray',$arr) 会调用 xmlToArray($arr)这个函数

dump() 人性化的打印输出 可以为 array|string|int|object|bool只需要一个参数即可。

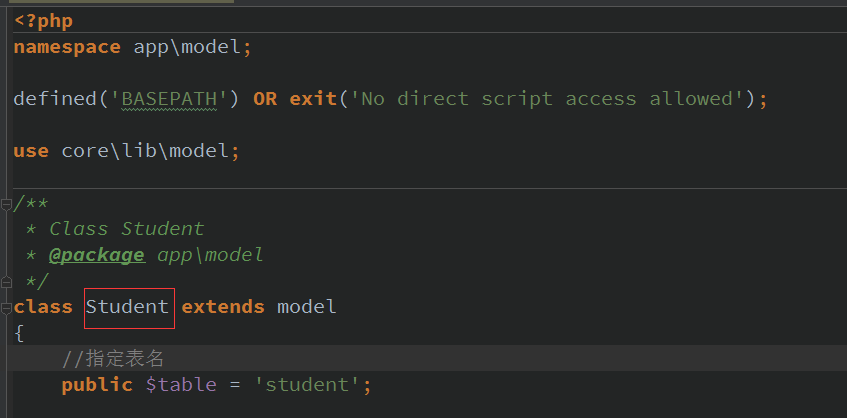
模型详解：

建立模型时,文件命名规范请使用驼峰法,类名要与文件名相同,例如文件名叫Student.php 类名必须是 Student

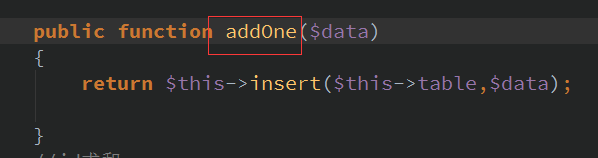
命名空间根据自身的情况默认app\model; 如果有需求要操作数据库 请继承 core\lib\model父类 或者use core\lib\model; 然后可以直接继承model即可 数据库配置文件在app/config/databases.php中 请根据实际情况进行修改

注意点--> 因为model父类的方法都是靠PDO来支持的 所以必须要开启PDO扩展哦

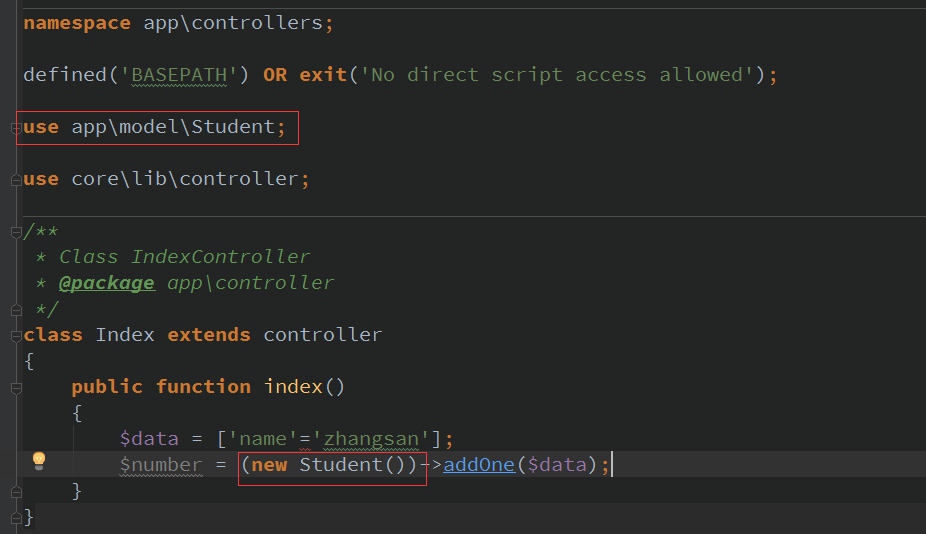
参考:



控制器中如何调用模型中的方法？ 很简单。 假如要在index控制器中调用Student模型中的addOne() ;



首先是实例化这个模型类的时候要命名空间引入进来 然后可以直接实例化这个类 在去调用其中的方法。示例一:



示例二:也可以不引入命名空间 直接去实例化所在的空间里面的类调用这个类里面的方法 如果多次要使用这个类 这样写不太方便 建议使用第一种方法 具体实现如下:



父类模型方法详解

select($table, $join, $columns, $where) 查询

表名.

table [string]

多表查询,不使用可以忽略.

join [array]

要查询的字段名.

columns [string/array] 你可以使用\*来匹配所有字段, 但如果你指名字段名可以很好的提高性能. 你可以使用别名，以防止字段冲突 如

$data = $database->select("account", [

"user\_id",

"nickname(my\_nickname)" //括号内代表字段别名

], [

"LIMIT" => 20

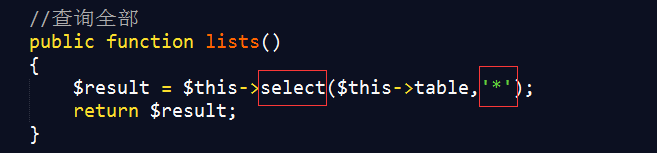
]);

where (optional) [array]

查询的条件.

Reutrn 返回是一个二维数组

示例查询某个表的全部数据 使用\*代替所有字段



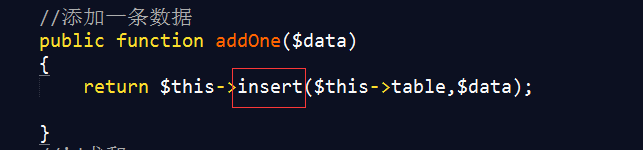
insert($table, $data) 插入

$table 表名

$data 数组 要插入的表字段和对应的值 多条数据可以使用三维数组

return 主键Id

示例:



update($table, $data, $where) 修改

table [string]

表名.

data [array]

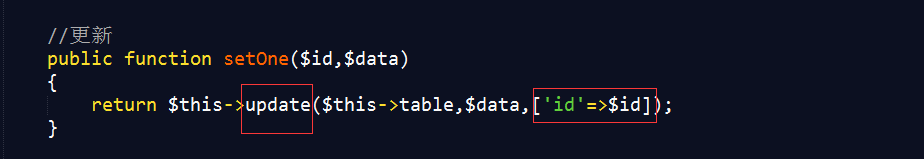
修改的数据.

where (optional) [array]

WHERE 条件 条件一定要存在 不然会修改整个表的数据

return 返回受影响的行数

示例:



delete($table, $where) 删除表中的数据 一个比较危险的操作，操作前请考虑好。。。

$database->delete($database->table,['id'=>$id]);

多个条件 请使用

$database->delete("table", [

"AND" => [

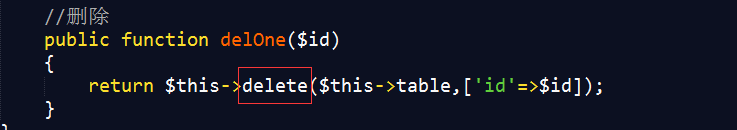
"type" => "business"

"age[<]" => 18

]

]);

Return: [int] 返回被删除的行数.



$where 条件详解 //后 代表where条件 原生sql

$database->select("account", "user\_name", [

"email" => "foo@bar.com"

]);

// WHERE email = 'foo@bar.com'

$database->select("account", "user\_name", [

"user\_id" => 200

]);

// WHERE user\_id = 200

$database->select("account", "user\_name", [

"user\_id[>]" => 200

]);

// WHERE user\_id > 200

$database->select("account", "user\_name", [

"user\_id[>=]" => 200

]);

// WHERE user\_id >= 200

$database->select("account", "user\_name", [

"user\_id[!]" => 200

]);

// WHERE user\_id != 200

$database->select("account", "user\_name", [

"age[<>]" => [200, 500]

]);

// WHERE age BETWEEN 200 AND 500

$database->select("account", "user\_name", [

"age[><]" => [200, 500]

]);

// WHERE age NOT BETWEEN 200 AND 500

// 你不仅可以使用字符串和数字，还可以使用数组

$database->select("account", "user\_name", [

"OR" => [

"user\_id" => [2, 123, 234, 54],

"email" => ["foo@bar.com", "cat@dog.com", "admin@medoo.in"]

]

]);

// WHERE

// user\_id IN (2,123,234,54) OR

// email IN ('foo@bar.com','cat@dog.com','admin@medoo.in')

// 多条件查询

$database->select("account", "user\_name", [

"AND" => [

"user\_name[!]" => "foo",

"user\_id[!]" => 1024,

"email[!]" => ["foo@bar.com", "cat@dog.com", "admin@medoo.in"],

"city[!]" => null,

"promoted[!]" => true

]

]);

//条件搜索

//你可以使用"AND" 或 "OR" 来拼接非常复杂的SQL语句

// 基础使用

$database->select("account", "user\_name", [

"AND" => [

"user\_id[>]" => 200,

"age[<>]" => [18, 25],

"gender" => "female"

]

]);

// WHERE user\_id > 200 AND age BETWEEN 18 AND 25 AND gender = 'female'

$database->select("account", "user\_name", [

"OR" => [

"user\_id[>]" => 200,

"age[<>]" => [18, 25],

"gender" => "female"

]

]);

// WHERE user\_id > 200 OR age BETWEEN 18 AND 25 OR gender = 'female'

// 复合条件

$database->has("account", [

"AND" => [

"OR" => [

"user\_name" => "foo",

"email" => "foo@bar.com"

],

"password" => "12345"

]

]);

// WHERE (user\_name = 'foo' OR email = 'foo@bar.com') AND password = '12345'

// 模糊匹配 Like

// LIKE 使用语法 [~]

// 默认情况下，使用%在前后包含关键词

$database->select("person", "id", [

"city[~]" => "lon"

]);

WHERE "city" LIKE '%lon%'

// 数组形式，查询多个关键词

$database->select("person", "id", [

"city[~]" => ["lon", "foo", "bar"]

]);

WHERE "city" LIKE '%lon%' OR "city" LIKE '%foo%' OR "city" LIKE '%bar%'

// 不包含 [!~]

$database->select("person", "id", [

"city[!~]" => "lon"

]);

WHERE "city" NOT LIKE '%lon%'

排序

$database->select("account", "user\_id", [

"ORDER" => "user\_id",

"ORDER" => [

"user\_id" => [43, 12, 57, 98, 144, 1],

"register\_date",

"profile\_id" => "DESC",

"date" => "ASC"

]

]);

使用SQL函数

在一些特殊的情况下，你可能需要使用SQL系统函数，只需要字段名前加上#号即可

$data = $database->select('account', [

'user\_id',

'user\_name'

], [

'#datetime' => 'NOW()'

]);

其它参数

$database->select("account", "user\_id", [

"GROUP" => "type",

"HAVING" => [

"user\_id[>]" => 500

],

"LIMIT" => [20, 100]

]);

// SELECT user\_id FROM account

// GROUP BY type

// HAVING user\_id > 500

// LIMIT 20,100

get($table, $columns, $where) 从表中返回一行数据

$email = $database->get("account", "email", [

"user\_id" => 1234

]);

Return: [string/array] 返回查询到的数据.

Has() 确定数据是否存在 has($table, $where)

$database->has("account", [

"AND" => [

"OR" => [

"user\_name" => "foo",

"email" => "foo"

],

"password" => "12345"

]

])

Return: [boolean] 返回 TRUE 或者 FALSE.

count($table, $where) 获取数据表中的行数

table [string]

表名.

where (optional) [array]

WHERE 条件.

$database->count($database->table);

Return: [number] 行的数量.

max($table, $column, $where) 获得数据表中，值最大的

table [string]

表名.

column [string]

查询的字段列.

where (optional) [array]

WHERE 条件.

$database->max($database->table,'id');

Return: [number] 返回最大的值.

min($table, $column, $where) 获得某个列中的最小的值

table [string]

表名.

column [string]

需要查询的列.

where (optional) [array]

WHERE 条件.

$database->min($database->table,'id');

Return: [number] 返回最小的值.

avg($table, $column, $where) 获得某个列字段的平均值

table [string]

表名.

column [string]

列字段

where (optional) [array]

WHERE 条件.

$database->avg($database->table,'id');

Return: [number] 平均值.

sum($table, $column, $where) 某个列字段相加

table [string]

表名.

column [string]

查询的列.

where (optional) [array]

WHERE 条件.

$database->sum($database->table,'id');

Return: [number] 相加的值.

replace($table, $columns, $where) 将新的数据替换旧的数据

$database->replace("account", "type", [

"user" => "new\_user",

"business" => "new\_business"

], [

"user\_id[>]" => 1000

]);

// Return: [int] 返回受影响的行数

query($query) 执行sql语句

query [string]

Return: [object] The PDOStatement object.

此方法用于使用复杂的SQL语句，SQL注入问题请参考quote方法

$database->query("CREATE TABLE table (

c1 INT STORAGE DISK,

c2 INT STORAGE MEMORY

) ENGINE NDB;");

$data = $database->query("SELECT email FROM account")->fetchAll();

print\_r($data);

quote($string) 字符串转义

$string [string]

字符串.

Return: [string]

$data = "Medoo";

echo "We love " . $data; // We love Medoo

echo "We love " . $database->quote($data); // We love 'Medoo'

PDO

$database->pdo->beginTransaction();//开启事务

$database->insert("account", [

"user\_name" => "foo",

"email" => "foo@bar.com",

"age" => 25

]);

$database->pdo->commit(); //提交

$database->pdo->rollBack(); //回滚

Debug()

输入sql语句，但不执行

输出sql语句，不需要使用echo或其它方法。调试完成请移除此代码

$database->debug()->select("bccount", [

"user\_name",

"email"

], [

"user\_id[<]" => 20

]);

获得最后一个执行的错误.

error()

Return: [array] 返回错误的数组代码

$database->select("bccount", [

"user\_name",

"email"

], [

"user\_id[<]" => 20

]);

var\_dump($database->error());

// array(3) { [0]=> string(5) "42S02" [1]=> int(1146) [2]=> string(36) "Table 'my\_database.bccount' doesn't exist" } /如果定义一个不存在的表名 则会说 表不存在

log() 返回所有执行的查询。 可以用此函数来查询、调试SQL语句

$database->insert("account", [

"user\_name" => "foo",

"email" => "foo@bar.com"

]);

var\_dump( $database->log() );

// array(1) {

// [1]=> string(74) "INSERT INTO "account" ("user\_name", "email") VALUES ('foo', 'foo@bar.com')"

// }

Last\_query() 返回最后一条执行的SQL语句. return: [string]

和log()很相似，但只返回最后一次执行的SQL语句

info() 查看数据库相关信息

Return: [string] 返回数据库相关信息

print\_r($database->info());

/\*

Array

(

[server] => Uptime: 5074 Threads: 1 Questions: 15 Slow queries: 0 Opens: 67 Flush tables: 1

Open tables: 60 Queries per second avg: 0.002

[client] => mysqlnd 5.0.10 - 20111026 - $Id: e707c415db32080b3752b232487a435ee0372157 $

[driver] => mysql

[version] => 5.6.10

[connection] => localhost via TCP/IP

)