## tp5生命周期

和php其他框架一样入口文件在public/index.php

## 2.路由

## 3.数据库

### 查询构造器

#### 原生查询

##### query方法

用于执行SQL查询操作，如果数据非法或者查询错误则返回false ,否则返回查询结果集(同select()方法 ，select方法查询结果不存在，返回空数组)

Db:query(‘select \* from think\_user where status=1’);

如果你当前采用了分布式数据库，并且设置了读写分离的话，query方法始终是在读服务器执行，因此query方法对应的都是读操作，而不管你的SQL语句是什么。

##### execute方法

用于更新和写入数据的sql操作，如果数据非法或者查询错误则返回false,否则返回影响的记录数

Db::execute(“update think\_user set name=’think’ where status=’1’”);

Db::execute(“insert into think\_user (‘name’,’age’) values(‘SWE’,18)”);

如果你当前采用了分布式数据库，并且设置了读写分离的话，execute方法始终是在写服务器执行，因此execute方法对应的都是写操作，而不管你的SQL语句是什么

##### 参数绑定

支持在原生查询的时候使用参数绑定，包括问号占位符或者命名占位符

Db:query(“select \* from think\_user where id=? And status =?”,[8,1]);

//命名绑定

Db::execute(“update think\_user set name=:name,age=:age where status:status ”,

[‘name’=>’swe’,’age’=>19,’status’=>’1’]);

#### 查询数据

##### 基本查询

Db::table(‘think\_user’)->where(‘status’,1)->find();

Find方法查询结果不存在，返回null

Db::table(‘think\_user’)->where(‘status’,1)->select();

Select方法查询结果不存在返回空数组

默认情况下find和select方法放回的都是数组

如果设置了表前缀可以使用,如果没有使用表前缀name方法和table方法一样

Db::name (‘think\_user’)->where(‘status’,1)->find();

Db::name(‘think\_user’)->where(‘status’,1)->select();

##### 助手函数

系统提供了一个系统函数，可以方便查询

db('user')->where('id',1)->find();

db('user')->where('status',1)->select();

##### 这部分没仔细看

注意：使用db助手函数默认每次都会重新连接数据库，而使用Db::name或者Db::table方法的话都是单例的。db函数如果需要采用相同的链接，可以传入第三个参数，例如：

db('user',[],false)->where('id',1)->find();

db('user',[],false)->where('status',1)->select();

上面的方式会使用同一个数据库连接，第二个参数为数据库的连接参数，留空表示采用数据库配置文件的配置。

##### 使用query对象或闭包对象

$query = new \think\db\Query();  
$query->table('think\_user')->where('status',1);  
Db::find($query);  
Db::select($query);

直接使用闭包函数查询

Db::select(function($query){

$query->table(‘think\_user’)->where(‘status’,1);

})

## 4.模型

### 更新

#### 直接更新数据

$user = new User;

// save方法第二个参数为更新条件

$user->save([

'name' => 'thinkphp',

'email' => 'thinkphp@qq.com'

],['id' => 1]);

Save方法返回受影响的函数，没有就返回0，即为false

## session

### 赋值

// 赋值（当前作用域）

Session::set('name','thinkphp');

// 赋值think作用域

Session::set('name','thinkphp','think');

### 判断是否存在

// 判断（当前作用域）是否赋值

Session::has('name');

// 判断think作用域下面是否赋值

Session::has('name','think');

### 取值

// 取值（当前作用域）

Session::get('name');

// 取值think作用域

Session::get('name','think');

如果name不存在返回null

### 删除

//删当前作用域

Session::delete(‘name’);

//删除think作用域

Session::delete(‘name’,’think’);

### 指定当前作用域

Session::prefix(‘think’);

### 取值并删除

Session::pull(‘name’);

Name不存在返回null

### 清空

//清空当前作用域

Session::clear();

//清除think作用域

Session::clear(‘think’);

### 闪存数据，下次请求之前有效（v5.0.2+）

// 设置session 并且在下一次请求之前有效

Session::flash('name','value');

### 提前清除当前请求有效的数据（v5.0.2+）

// 清除当前请求有效的sessionSession::flush();

### 二级数组

支持seesion的二维数组操作

// 赋值（当前作用域）

Session::set('name.item','thinkphp');

// 判断（当前作用域）是否赋值

Session::has('name.item');

// 取值（当前作用域）

Session::get('name.item');

// 删除（当前作用域）

Session::delete('name.item');

### 助手函数

系统也提供了助手函数session完成相同的功能，例如：

// 初始化sessionsession([

'prefix' => 'module',

'type' => '',

'auto\_start' => true,

]);

// 赋值（当前作用域）

session('name', 'thinkphp');

// 赋值think作用域

session('name', 'thinkphp', 'think');

// 判断（当前作用域）是否赋值

session('?name');

// 取值（当前作用域）

session('name');

// 取值think作用域

session('name', '', 'think');

// 删除（当前作用域）

session('name', null);

// 清除session（当前作用域）

session(null);

// 清除think作用域

session(null, 'think');

## 验证器

## Request

## 助手函数

halt() 变量调试输出并中断执行

## 相对路径与绝对路劲

参考：<https://www.cnblogs.com/laijinquan/p/9783043.html>

使用  \_\_DIR\_\_  常量定义的路径都是相对路径，这个常量表示当前的路径，在不同地方使用，得出的结果都不一样。

你可以理解为除了使用这个常量之外定义的都是绝对路径（简单来说），一般来说使用  APP\_PATH      常量定义的路径都是绝对路径。

## Tp项目git上传下载文件缺失问题

Git公司的项目发现没有thinkphp目录，查阅资料得

参考：

<https://www.cnblogs.com/Renyi-Fan/p/8667320.html>

git项目的时候有一些文件不想上传可以使用gitignore

生成.gitignore文件   
在git bash中使用命令touch .gitignore，就可以在当前目录下生成.gitignore文件

编辑.gitignore文件   
在git bash中，使用vi .gitignore就可以编辑此文件，编辑结束后按Esc然后按:，最后输入wq表示保存退出，输入q表示不保存退出。这与在Linux终端中的操作是一样的。

## \_initialize() 区别 \_\_construct()

1、\_initialize（）方法是在任何方法执行之前，都要执行的，当然也包括\_ \_构造构造函数。也就是说如果存在\_initialize（）函数，调用对象的任何方法都会导致\_initialize（）函数的自动调用，

而\_construct构造函数仅仅在创建对象的时候调用一次，跟其它方法调用没有关系。

2、\_ \_construct这里是双划线，而\_initialize（）函数是单划线

3、如果父子类均有\_initialize （）函数，则子类覆盖了父类的，如果子类没有而父类有，则子类继承父类的。

4、默认情况下，子类的构造函数不会自动调用父类的构造函数。在调用子类对象的\_initialize（）时，也不会导致自动调用父类的\_initialize（）

5、实际编写子类的构造函数时，一般都要加上父类构造函数的主动调用parent::\_ \_construct()，否则会导致子类对象空指针的异常，如Call to a member function assign() on a non-object。