一、入口文件 index.php：

给入口文件添加一个 header 头声明字符集：

Header(‘Content-Type:text/html;charset=utf8’);

开启调试模式：

Define(‘APP_DEBUG’,true);

定义应用目录：

Define(‘APP_PATH’,’./Application/’);

定义一些常量：

```
//入口文件
//定义工作路径
define('WORKING_PATH',str_replace('\\','/',__DIR__));
//定义上传根目录
define('UPLOAD_PATH','/Public/Upload/');
```

引入 ThinkPHP 框架入口文件：

Require ‘./ThinkPHP/ThinkPHP.php’;

二、Controller(控制器)：

控制器的创建：

命名规则：控制器名+Controller+.class.php
控制器结构代码：

第一步：声明当前控制器的命名空间

Namespeace Home\Controller

第二步：引入父类控制器

Use ThinkPHP\Controller

第三步：声明控制器并继承父类

Class UserController extends Controller{
}

路由形式：

路由：是指访问项目中具体某个方法的 URL 地址
在 ThinkPHP 中系统提供了 4 种路由形式：

普通形式路由(get 形式路由)：

http://网址/入口文件?m=分组名&c=控制器&a=方法名&参数名=参数值

Pathinfo 形式路由(默认)：

http://网址/入口文件/分组名/控制器/方法名/参数名/参数值

Rewrite 形式路由：

http://网址/分组名/控制器/方法名/参数名/参数值

需要配置才能使用:

第一步：需要修改 Apache 配置文件 Apache/conf/httpd.conf，开启重写模块，去掉之前的注释符

LoadModule rewrite_module modules/mod_rewrite.so

第二步：需要修改虚拟主机配置文件 Apache/conf/extra/httpd-vhosts.conf

AllowOverride all

第三步：重新启动 Apache

第四步：将 ThinkPHP 压缩包中的.htaccess 复制到入口文件的同级目录(需注意该文件是不是需要的文件，因为运行模式文件有好几种)

兼容形式路由:

URL_MODEL 路由形式的配置值，不影响我们在地址栏中直接输入其他形式路由访问，影响的是 U 方法组装生成的 url 地址形式。

控制器中的跳转：

1.URL 组装

语法：U(‘url 路径’，参数数组);
例 1：要使用 U 方法组装出当前控制器下 Index 方法地址
U(‘index’);
例 2：要使用 U 方法组装出另一个控制器 Index 下 Index 方法地址
U(‘Index/index’);
例 3：要使用 U 方法给指定的页面添加参数，给 Index 控制器下的 index 方法传递一个 id=100
U(‘Index/index’,array(‘id’=>100,’name’=>’smith’));
总结出一个通用的格式：
U(‘[分组名/控制器/]方法名’，array(‘参数 1’=>’值 1’，‘参数 2’=>’值 2’));
URL_MODEL 路由形式的配置值，不影响我们在地址栏中直接输入其他形式路由访问，影响的是 U 方法组装生成的 url 地址形式。

2.系统跳转方法：

在 ThinkPHP 中系统有 2 个跳转方法，分别是成功跳转和失败跳转
成功：
$this->success(跳转提示，跳转地址，等待时间);
失败：
$this->error(跳转提示，跳转地址，等待时间);
跳转提示参数必须要有，不然会死循环。如果没有指定跳转地址，默认跳转到上一页。

空操作：

空操作是指系统在找不到指定的操作方法的时候，会定位到空操作方法来执行(针对控制器也是如此)，利用这个机制，我们可以实现错误页面和一些 URL 的优化。

关于空操作的说明：
1.空操作方法：在看控制器中可以定义一个操作方法名字叫做_empty()。创建一个 error.html 页面放到当前控制器对应的视图文件夹里。

```
public function _empty(){
    $this->display('error');
}
```

2.空操作控制器：在 ThinkPHP 中操作一个空的控制器，当指定的控制器找不到，则会去访问空的控制器，空控制器的文件名叫做 EmptyController.class.php。并且需要在 View 目录下中创建一个文件夹 Empty，然后把 404 页面 error.html 放入 Empty 文件夹，当控制器错误时调用_empty 方法，展示 error.html 页面。

```
class EmptyController extends Controller{
    public function _empty(){
        $this->display('Empty/error');
    }
}
```

在 ThinkPHP 中获取 ip 信息：

在 ThinkPHP 的系统函数库文件中封装了一个方法来获取 ip:get_client_ip()。

语法：

Get_client_ip(可选参数数字)

如果参数是 0 的话或者不写(默认)，则表示返回正常的 ipv4 地址

如果参数是 1 则表示返回 ipv4 地址对应的数字地址。

如何把 ip 地址转化成物理地址：

在 ThinkPHP 中系统提供了一个工具类，能实现转化，但是系统不提供转化所使用的数据，也就是说需要自己去寻找对应的数据库。数据库可以从纯真官网去寻找(http://www.cz88.net)。下载安装包 setup.zip 后安装。下载安装的程序后，在其安装目录中可以找到 qqwry.dat 文件，该危机就是后续所使用到的数据库文件。

ThinkPHP 提供的类：Org/Net/IPlocation.class.php

在实例化的时候可以传递一个文件名，文件名所在的位置和当前的类是同级目录，也就是需要把 qqwry.dat 文件复制到 Org/Net 目录下，和 Iplocation.class.php 在同一个目录。

Getlocation：

```
//接口写法
class TextController extends Controller{
    public function test(){
    //实例化对象
    $ip = new \Org\Net\IpLocation('qqwry.dat');
    $ss = I('get.ip');
    //查询
    $data = $ip ->getlocation($ss);
    dump($data);
    }
}
```

三、Model(模型)：

配置数据库连接：

index.php | ContentReplaceBehavior.class.php | config.php | config.php

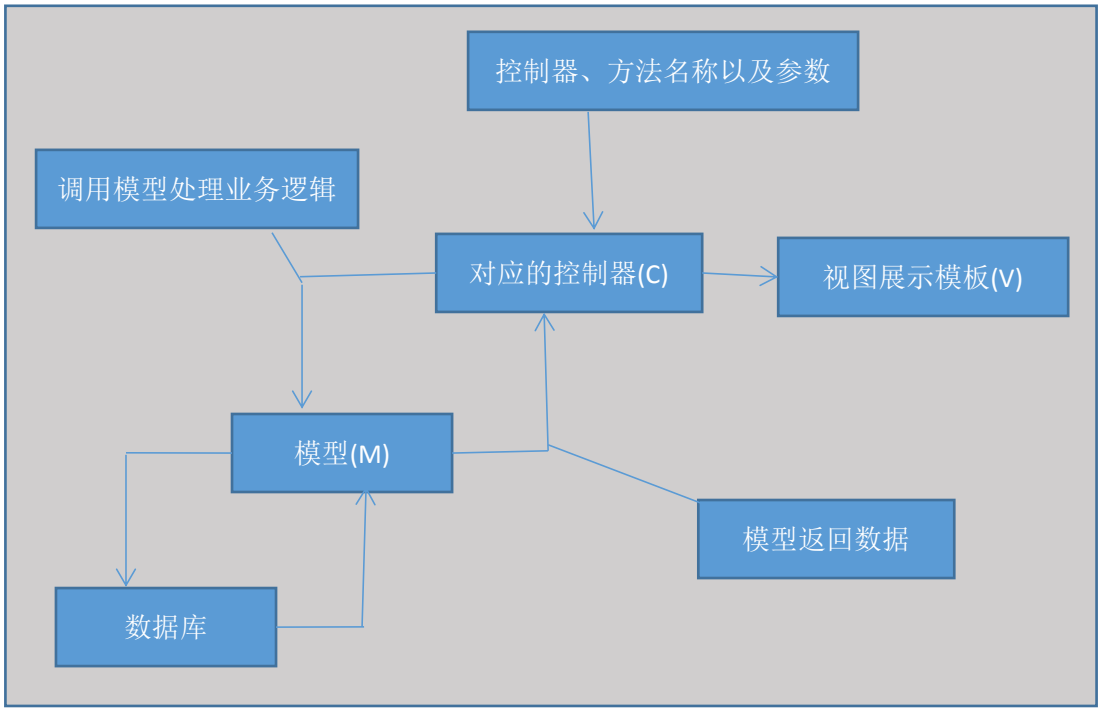
源 | 历史记录

```php
<?php
return array(
    'DB_TYPE'               =>  'mysql',     // 数据库类型
    'DB_HOST'               =>  '127.0.0.1', // 服务器地址
    'DB_NAME'               =>  'hrjkys',          // 数据库名
    'DB_USER'               =>  'root',      // 用户名
    'DB_PWD'                =>  'root',          // 密码
    'DB_PORT'               =>  '3306',        // 端口
    //开启页面Irace
    'SHOW_PAGE_IRACE' =>true,
);
```

不要修改 TP 框架的配置文件，把需要修改的内容复制后，在项目目录的配置文件里修改，如上。

模型的创建：

模型是 MVC 三大组成部分的 M(Model),作用是负责与数据库的数据交互(CURD)。



模型的创建：
命名规范：模型名(要求是不带前缀的表名并且首字母大写)+Model 关键词+.class.php
代码结构规范：

第一步：声明命名空间

第二步：引入父类模型 Model.class.php

第三部：声明模型并且继承父类模型

空模型也可以进行数据表的基本 CURD 操作，因为模型继承了父类，所以可以执行基本的操作。

模型的实例化：

模型创建完成后，由于模型本质是一个类，类在使用的时候需要实例化操作

普通实例化：

通过自己编写代码来 new 一个对象

```
$obj = new  类名();
```

在控制器中定义一个方法来实例化模型，使用的是普通方式实例化：

快速实例化方法：

TP 框架为了简单、快速、高效开发，提供了两个快速方法对模型实例化操作：
M()方法;

```
$obj = M(['不带前缀的表名']);
```

表达的含义：直接实例化父类模型(Model.class.php)。如果指定了表面，则实例化父类模型的时候管理指定的表，如果没有指定表面(没有传递参数)则不关联表，一般用于执行原生的 sql 语句(M()->query(原生 sql 语句))。
D()方法:

```
$obj = D(['模型名']);
```

表达的含义：实例化我们自己创建的模型(分组/Model 目录中)；如果传递了模型名，则实例化指定的模型，如果没有指定或者模型名不存在，则直接实例化父类模型(Model.class.php)。
D()和 M()方法的区别：D 方法是实例化自定义模型，如果自定义模型不存在，则实例化父类模型；M 方法本身就是直接实例化父类模型。两者的差异就是实例化的对象不一样。
在实际开发中，如果需要使用的操作在父类模型中已经封装好了，则可以直接实例化父类模型，使用 M 方法，如果父类中方法不能满足开发需求而需要自定义方法，则使用 D 方法实例化自定义模型。

增加操作：

在 TP 框架中封装好方法：

$model -> add(一维数组);

一维数组要求必须是键值(关联)数组，键必须和数据表中字段名要匹配，如果不匹配则在增加的时候会被 TP 过滤掉。

add 方法返回值是新增记录的主键 id。

如果需要添加多个记录：

1.可以循环使用 add 方法;

2.使用 addAll 方法，语法：$model->addAll(二维数组)，这里的二维数组也必须是关联数组。外层数组必须是下标从 0 开始的连续索引数组。

修改操作：

在 TP 框架封装好了方法：

$model->save(一维关联数组);

条件需要一维关联数组必须要有主键信息。如果没有主键信息，则相当于批量修改，这在 TP 中是不允许的，目的是为了防止误操作。

如果没有指定主键信息，返回值是 false,表示修改操作并没有执行，而不是指 Mysql 执行失败。

正常执行的返回值：表示受到影响的行数。int(1)

查询操作：

在 TP 中封装好了方法：select 方法和 find 方法

select 方法语法：

$model->select();	表示查询全部的信息
$model->select(id);	表示查询指定 id 的信息
$model->select('id1,id2,id3...');	表示查询指定 Id 集合的信息。

find()方法语法：

$model->find();	表示查询当前表中第一条记录
$model->find(id);	表示查询表中指定 id 的记录

返回值：

select 方法返回值是一个二维数组，即使查询的是一条记录返回也是二维数组；find 返回值是一维数组。

删除操作：

在 TP 框架中封装好了一个方法：delete();

$model->delete();	不能使用，删除方法必须有条件

$model->delete(id);　　删除指定 id 对应的记录

$model->delete('id1,id2,id3...');　　删除多个 id 对应的记录

说明：

删除分为两种删除：

物理删除：真删除

逻辑删除：假删除，本质是修改操作。在数据表中定义一个状态字段，比如说是 status,取值是 0 或者 1，在查询的时候读取状态是 1 的；当用户点击删除之后触发修改操作，将状态从 1 修改成 0。之后状态是 0 的就不会被显示。

四、View(视图):

什么是视图：

视图就是 MVC 三大组成部分中 V(view),主要负责信息的输出和展示

视图的创建：

创建的位置需要是在分组目录下的 View 目录下与控制器同名的目录中 View/Index/index

视图的展示：

在 smarty 中展示模板使用 display 方法，在 ThinkPHP 中同样也是 display 方法
语法格式：

$this->display(); 展示当前控制器下与当前请求方法名称一致的模板文件

$this->display('模板文件名[不带后缀]'); 展示当前控制器下的指定模板文件

$this->display('View 目录下的目录名/模板文件名[不带后缀]'); 展示指定控制器目录下的指定模板文件

变量分配(初阶)：

在实际开发中不仅仅是展示模板，往往还需要数据的输出。这个时候变量还在控制器的方法中，需要将数据给传递到模板中并且展示，这个过程叫做变量分配。
在 ThinkPHPzhong 系统封装好了一个变量的分配方法，这个方法叫做 assign()方法。语法如下：

$this->assign('模板中变量名',$php 中的变量名);

一般情况，两个参数的变量名都是一样的。
目前已经将 var 变量传递给了模板文件，在模板文件中展示数据如下：

{$模板中变量名}

例如：

```
$var = date('Y-m-d H:i:s',time());
$this->assign('var',$var);
$this->display();
```

默认的变量左右分隔符是{和}，是可以修改的。在配置文件中找到具体配置项 TMPL_L_DELIM 和 TMPL_R_DELIM。

变量分配(进阶)：

一维数组的输出：

在 ThinkPHP 中变量的分配都是使用 assign 方法。

```
class TestController extends Controller{
    function test(){
        $array = array(
            '西游记','三国演义','水浒传','红楼梦'
        );
        $this ->assign('array',$array);
        $this->display();
    }
}
```

在 php 中输出数组的具体元素可以通过下标的形式输出：$array[key]
关于数组在模板中输出的语法格式：

支持中括号形式：{$array[key]}

点形式：{$array.key}

定义模板并且输出：

```
    <body>
        中括号形式：{$array[0]}
        点形式：{$array.0}
    </body>
```

两种形式在输出的效果上没有什么区别，根据自己的喜好使用。

二位数组的输出：

在方法中定义一个二维数组，并分配到模板。

```
    //二维数组
    function test2(){
        $array2 = array(
            array('大师兄','二师兄','三师兄'),
            array('宋江','林冲','李逵'),
            array('贾宝玉','林黛玉','薛宝钗'),
        );
        $this ->assign('array2',$array2);
        $this->display();
    }
}
```

关于数组在模板中输出的语法格式：

支持中括号形式：{$array[key1][key2]}
点形式：{$array.key1.key2}
两种形式在输出的效果上没有什么区别，根据自己的喜好使用。

定义模板并且输出：

```
<body>
    中括号形式：{$array[0][1]}
    点形式：{$array.0.1}
</body>
```

对象变量的输出：

对象在实例化之后一般会保持到一个变量中，这个变量也可以被分配到模板当中去。

```
//定义一个类对象
namespace Admin\Controller;

class Student{

}
```

实例化对象：

```
    //变量分配(对象)
    function test3(){
        //实例化student对象，student是一个类
        $stu = new Student();
        $stu->id=100;
        $stu->name='小明';
        $stu->age=18;

        $this->assign('stu',$stu);
        $this->display();
    }
}
```

在 php 中通过$obj->attr 或者$obj::attr 形式输出对象的属性.
在 ThinkPHP 的模板中输出属性的值，可以通过下面的两种方式来实现：
支持箭头形式：{$obj->attr}
支持冒号形式：{$obj::attr}

```
<body>
    箭头形式：{$stu->id}
    冒号形式：{$stu:id}
</body>
```

切记：在模板中输出对象的属性时，不能使用点形式。

系统变量：

在 ThinkPHP 中系统提供了一下几个系统变量(超全局变量在模板中的使用)：

$Think.server	等价于$_SERVER,获取服务器的相关信息
$Think.get	等价于$_GET,获取 get 请求的信息
$Think.post	等价于$_POST,获取 post 请求的信息
$Think.request	等价于$_REGUEST,获取 get 和 post 请求的信息
$Think.cookie	等价于$_COOKIE,获取 cookie 中的信息
$Think.session	等价于$_SESSION 获取 session 中的信息
$Think.config	获取 ThinkPHP 中所有配置文件的一个综合，如果后面指定了元素，则获取指定的配置。

语法格式：

{$Think.xxx.具体的元素下标}

例如：

获取 get 请求中的 id

{$Think.get.id}

案例：

{$Think.server.path}

{$Think.get.id}

{$Think.post.id}

{$Think.request.id}

{$Think.config.DEFAULT_MODULE}

视图中使用函数：

在实际开发中，有些变量在模板中不能直接使用，比如在数据表中存储的时间戳，必须在视图中使用函数的方式来解决这个问题：

语法格式：

{$var|fun1|fun2=参数 1，参数 2...}

参数说明：

$var ：	变量
|：	变量修饰符
Fun1:	表示需要使用的第一个函数
Fun2:	表示需要使用的第二个函数
参数 1、参数 2：	函数 2 的参数

=：表示该函数有参数

###:表示变量的自身

案例 1：时间戳的格式化。

```php
class TestContorller extends Controller{
    public function test1(){
        //定义时间戳
        $time = time();
        //传递参数给模板
        $this -> assign('time',$time);
        //展示模板
        $this -> display();
    }
}
```

模板中应用：

```
<body>
{$time|date='Y-m-d H:i:s ',###}
</body>
```

输出结果：

2017-3-14　15:10:22

特别说明：

第一：###什么时候该写，什么时候不该写。当需要使用的函数只有一个参数并且是变量自身的时候，###可以省略。当需要使用的函数有多个参数，但是其第一个参数是变量自身的时候，也可以省略不写###。

第二：关于函数名的说明，函数名对应的函数必须是 php 内置的函数或者是在函数库文件中定义好的函数，其他主观臆造的函数不能使用。

案例 2：定义一个字符串，截取其中的前 5 个字符，并将其转化为大写。

```php
class TestContorller extends Controller{
    public function test(){
        //定义字符串
        $str = 'akKULdsdUliDlw';
        //传递参数给模板
        $this -> assign('str',$str);
        //展示模板
        $this -> display();
    }
}
```

分析：截取字符串使用函数 substr,转化大写使用 strtoupper。

Substr(变量，起始位置，长度);

模板中使用：

```
<body>
{$str|strtoupper|substr=0,5}
{$str|substr=0,5|strtoupper}
</body>
```

这里的 substr 函数由于第一个参数是变量$str 本身，所以###省略不写，写全因该是{$str|substr=###,0,5}

默认值：

默认值：就是当某个变量不存在或者为空的时候，就会显示默认的字符，默认的字符就是变量的默认值。
语法：

{$变量名|default=默认值}

Default 是 TP 中封装的一个函数，默认值是函数的参数。

运算符：

{$a+$b}
{$a-$b}
{$a*$b}
{$a/$b}
{$a%$b}
{$a++}或{++$a}
{$a--}或{--$a}

在方法中定义变量 a 和变量 b 传递给模板，在模板中运算。

文件引入/包含：

在实际开发的时候一般情况下会把网站的公共部分，如头部、尾部等可以单独存放到一个文件中，在后期的时候可以直接引入该部分。维护方便，代码不重复。
在 TP 中系统提供了一个模板标签，可以让我们引入一些公共部分的代码文件，这个标签是 include 标签：

<include file='需要引入的模板文件'>

说明：路径可以是相对路径，但是相对于入口文件的。

```
<body>
    <include file="Application/Admin/View/Test/head.html" />
    <div>中间内容部分</div>
    <include file="Application/Admin/View/Test/foot.html" />
</body>
```

说明：在实际开发的时候，上述的路径很长不容易记，一般采用另外一种比较简单的方法：

<include file='View 目录名/模板文件名' />

```
<body>
    <include file='Test/head.html' />
    <div>中间内容部分</div>
    <include file='Test/foot.html' />
</body>
```

除了使用 include 标签来引入文件外，include 标签还有另外一个用法：传递参数给引入文件

<include file='文件路径' 参数名='参数值' />

```
<body>
    <include file='Test/head.html' />
    <div>中间内容部分</div>
    <include file='Test/foot.html' title='2017年' />
</body>
```

在目标文件中使用参数：

[参数名]

```
<body>
    今年是[title]
</body>
```

说明：如果目标文件中的参数[title]不存在，则[title]会被原样输出到浏览器上。

循环遍历：

在 TP 中系统提供了 2 个标签来实现数组在模板中的遍历：

Volist 标签：

<volist name='需要遍历的模板变量' id='当前遍历到的元素'>
</volist>
案例 1：一维数组

```
class TestContorller extends Controller{
    public function test(){
        //定义一个一维数组
        $array = array('元素1','元素2','元素3');
        //传递参数给模板
        $this -> assign('array',$array);
        //展示模板
        $this -> display();
    }
}
```

模板中遍历一维数组：

```
<body>
    <volist name='array' id='vo'>
        {$vo}
    </volist>
</body>
```

案例 2：二维数组

```
class TestContorller extends Controller{
    public function test(){
        //定义一个一维数组
        $array = array(
            array('元素1','元素2','元素3'),
            array('项目1','项目2','项目3'),
            array('属性1','属性2','属性3')
        );
        //传递参数给模板
        $this -> assign('array',$array);
        //展示模板
        $this -> display();
    }
}
```

模板中遍历二维数组：

```
<body>
    <volist name='array' id='vol'>
        <volist name='vol' id='vo'>
            {$vo}
        </volist>
    </volist>
</body>
```

foreach 标签：

<foreach name='需要遍历的模板变量' item='当前遍历到的元素'>
</foreach>

两个标签大体上是一样的，区别在于：volist 除了上述的 name 和 Id 属性外，还支持更多的属性对，如 Mod、key、length 等。而 foreach 除了上述的 name 和 item 外只支持 key 属性对，可以理解为 foreach 标签是 volist 标签的一个简化版本。建议使用 volist 标签。

IF 标签：

语法格式：

<if condition=’条件 1’>
输出结果 1
<elseif condition=’条件 2’>
输出结果 2
<else/>
最后一个输出结果
</if>

案例：输出今天的星期，传递给模板

```
class TestController extends Controller{
    public function test(){
        //输出今天的星期数字
        $day = date('N',time());
        //传递参数给模板
        $this -> assign('day',$day);
        //展示模板
        $this -> display();
    }
}
```

在模板中使用：

```
<body>
{$day}
<if condition='$day == 1'>
    星期一
    <elseif condition='$day == 2' />
    星期二
    <elseif condition='$day == 3' />
    星期三
    <elseif condition='$day == 4' />
    星期四
    <elseif condition='$day == 5' />
    星期五
    <elseif condition='$day == 6' />
    星期六
    <else/>
    星期天
</if>
</body>
```

Php 标签：

Php 标签即使指在模板中使用 php 语句。
TP 支持 2 中形式：
　　第一种：PHP 内置的 php 标签；语法格式<?php php 代码?>
　　第二种：TP 内置的 php 标签;语法格式<php>php 代码<php>
在世界开发的时候一般情况下不建议在模板中使用 php 标签。在配置项中有一个配置项可以禁用 php 标签，配置项叫做：TMPL_DENY_PHP,只是禁用 php 原生代码，并不会禁用 TP 封装的 php 标签。默认是 false，禁用改为 true。而且禁用后，会出错。建议不禁用。

模板常量替换机制：

在实际开发中会出现一个这样的问题：在引入图片、CSS 和 JS 文件的时候，往往需要写一些比较复杂的路径，这个时候可以考虑使用模板常量替换的机制。
在 ThinkPHP 中系统默认提供了几个常用的模板常量(模板常量不能在控制器中使用)。
　　__MODULE__　　表示从域名后面开始一直到分组名结束的路由
　　__CONTROLLER__　表示从域名后面开始一直到控制器结束的路由
　　__ACTION__　　表示从域名后面开始一直到方法名结束的路由
　　__PUBLIC__　　站点根目录下的 Public 目录的路由
　　__SELF__　　表示从域名后面开始一直到路由的最后(如果没有参数则和__ACTION__所输出的内容是一样的)。
在 ThinkPHP 中“模板常量”是通过模板内容替换机制来实现的，并非是常量的定义，替换机制可以查看行为文件 ControllerReplace.Behaviour.class.php

```
protected function templateContentReplace($content) {
    // 系统默认的特殊变量替换
    $replace =  array(
        '__ROOT__'       =>  __ROOT__,       // 当前网站地址
        '__APP__'        =>  __APP__,        // 当前应用地址
        '__MODULE__'     =>  __MODULE__,
        '__ACTION__'     =>  __ACTION__,     // 当前操作地址
        '__SELF__'       =>  htmlentities(__SELF__),      // 当前页面地址
        '__CONTROLLER__' =>  __CONTROLLER__,
        '__URL__'        =>  __CONTROLLER__,
        '__PUBLIC__'     =>  __ROOT__.'/Public',// 站点公共目录
    );
    // 允许用户自定义模板的字符串替换
    if(is_array(C('TMPL_PARSE_STRING')) )
        $replace =  array_merge($replace,C('TMPL_PARSE_STRING'));
    $content = str_replace(array_keys($replace),array_values($replace),$content);
    return $content;
}
```

其模板常量的实现核心就是字符串的替换。
为了后期使用的方便，我们可以在配置文件中定义模板常量。

模板内容获取方法：

在 ThinkPHP 中有一个方法和 display()有点相似，叫做 fetch 方法

Display()方法：$this->display()　展示模板

Fetch()方法:$this->fetch();　获取模板(有返回值)

在底层实现上的差异：

Display()方法：替换模板中常量/变量->获取模板内容->输出模板内容

Fetch()方法：替换模板中常量/变量->获取模板内容

Display()方法的前 2 步操作实际上是通过 fetch 方法实现的。

例如；

```
$str = $this->fetch();
Echo $str;
```

视图中的注释：

视图中的注释特指 ThinkPHP 中视图的注释。

Html 中的注释<!--注释内容-->，不会在网页中显示，但是会在页面的源代码中被输出。

ThinkPHP 中的模板注释：

行注释：{//行注释内容}

块注释：{/*块注释内容*/}

ThinkPHP 中的模板注释内容，不会在网页中显示，也不会在页面的源代码中被输出。

普通的 html 注释属于客户端注释，会在浏览器的源代码中输出；而 ThinkPHP 中的模板注释则属于服务器端的注释，不会被浏览器输出。

注意：行注释不要当作块注释来写，在行注释和块注释当中(大括号里面)不要再出现大括号(模板变量)

五、扩展

Ueditor 富文本编辑器：

Ueditor 是百度公司开发的，还有一款类似的插件，叫做 CKeditor。
上百度搜索 Ueditor 下载 PHP 版本的压缩包，使用可以参考 demo.html。
使用 UE 的步骤：

第一步：引入外部的资源文件(javascript 文件)，实际应用中需要修改路径

Ueditor.config.js

Ueditor.all.min.js

Zh-cn.js

```
<script type="text/javascript" charset="utf-8" src="ueditor.config.js"></script>
<script type="text/javascript" charset="utf-8" src="ueditor.all.min.js"> </script>
<!--建议手动加在语言，避免在ie下有时因为加载语言失败导致编辑器加载失败-->
<!--这里加载的语言文件会覆盖你在配置项目里添加的语言类型，比如你在配置项目里配置的是英文，这里
<script type="text/javascript" charset="utf-8" src="lang/zh-cn/zh-cn.js"></script>
```

第二步：指定标签，设置容器的位置(编辑器显示的位置)

```
<div>
    <h1>完整demo</h1>
    <script id="editor" type="text/plain" style="width:1024px;height:500px;"></script>
</div>
```

上述代码是通过 id 来指定容器的名字，后期实例化容器的时候需要指定的 Id。

第三步：实例化容器，生成编辑器效果。

```
<script type="text/javascript">

    //实例化编辑器
    //建议使用工厂方法getEditor创建和引用
    var ue = UE.getEditor('editor');
```

使用 UE 编辑器替换掉视图文件中的 textarea。

第一步，将 utf8-php 解压目录复制到公共静态资源文件目录下的插件目录(plugin)

第二步，在目标模板文件中引入三个需要的 js 文件。

第三步，设置标签，定义富文本编辑器的位置。

第四步，实例化容器

问题：

1.UE 编辑器默认的 name 值是 editorVlue，默认值和数据表中的字段名是不一样的，这就需要我们给当前容器的标签定义一个 name 值。

2.在 Ue 的源码中的一些样式会被 ThinkPHP 中的 I()方法进行转化成实体字符。

UE 使用说明：

1.防止 sql 注入和 xss：通过 I 方法是不能全部解决的，使用一个插件 htmlpurifiy 来对指定的标签进行过滤。

2.关于 UE 中的表情使用需要联网。

3.关于图片上传，该功能需要配置，配置文件 ue/php/config.json,需要指定路径

```
"imagePathFormat": "/ueditor/php/upload/image/{yyyy}{mm}{dd}/{time}{rand:6}",
```

在模板文件中读取数据库内容的时候需要将数据表中实体字符进行还原，可以使用函数 htmlspecialchars_decode。

验证码：

验证码：captcha(全自动识别机器与人类的图灵测试)。
常见的验证码分为三类：页面上的图片形式、短信验证码、语音验证码。
在 ThinkPHP 中，系统封装了一个验证码类：Verify.class.php
构造方法：__confstruct()在实例化的时候可以传递一个数组，用于和其他成员属性 config 进行合并，生成新的配置
check 方法：校验验证码，需要传递参数
entry:方法：输出图片，保持验证码到 session 中
步骤：

第一步：在控制器中创建 captcha 方法，用于输出验证码

```
public function captcha(){
    //配置
    $cfg = array(
        'fontSize' => 12,
        'useCurve' => false,
        'useNoise' => false,
        'length' => 4,
        'fontttf' => '4.ttf'
    );
    //实例化验证码类
    $verfiy = new \Think\Verify($cfg);
    //输出验证码
    $verfiy->entry();
}
```

第二步：在模板文件 login.html 中输出验证码，修改模板文件，调整大小和样式，增加点击事件刷新验证码。

```
<form action="{:U('checklogin')}" method="post">
<img src="__CONTROLLER__/captcha" onclick="this.src='__CONTROLLER__/captcha/t/'+Math.random()"/>
</form>
```

第三步：<a>连接提交的话，需要修改 Jquery 代码段

```
<script>
    $('.btn').on('click',function(){
        $('form').submit();
    })
</script>
```

第四步：编写 checklogin 方法用于处理用户登录

先验证验证码是否正确，如果正确再验证用户名和密码。

```
public function checkLogin(){
    //接受数据
    $post = I('post.');
     //实例化验证码类
    $verfiy = new \Think\Verify($cfg);
    //验证
    $result = $verfiy->check($post['captcha']);
    if($result){
        //验证码正确
        $model = M('User');
        //删除验证码元素，不需要在数据库验证
        unset($post['captcha']);
        //查询
        $data = $model->where($post) ->find();
        //判断是否存在用户
        if($data){
            //用户存在，用户信息保存到session中，跳转页面
            session('id',$data['id']);
            session('username',$data['username']);
            session('role_id',$data['role_id']);
            $this->success('登录成功',U('Index/index'),3);
        }else{
            //用户名或密码错误
            $this->error('用户名或密码错误');
        }
    }else{
        //验证码错误
        $this->error('验证码错误');
    }
```

退出登录：

```
//退出方法

public function logout(){
    session(null);
    $this->success('退出成功',U('login'),3);
}
```

关于中文验证码的几点说明：

第一：不到万不得已的情况下不要使用中文验证码

第二：中文验证码需要中文字体的支持，中文字体可以在自己的计算机中找到。

第三：使用中文验证码必须开启 php 的扩张 mbstring

上传类：

ThinkPHP 中系统封装了一个上传类：Upload.class.php

方法:

构造方法：可以在实例化的时候传递一个配置数组，有内部进行合并配置操作

getError：用户获取最后一次上传的错误信息。

$upload->getError();

uploadone:上传单个文件

参数是$_FILES 中的子元素，返回值是上传的结果。成功返回的时候具有九个元素的一维数组，失败返回 false。

upload:上传多个文件，参数通常是$__FILES 整个数组，成功返回上传的结果，是二维数组。失败返回 false。

注意点：

第一：表单必须有 enctype=”multipart/form-data”属性

第二：表单中的文件域 type 类型必须是 file

第三：表单提交方式必须是 Post。

案例：

第一步：控制器里创建 add 方法，调用模型里的 saveDate 方法保存数据

为了符合 MVC 的设计规范，我们需要自定义一个模型，然后将文件上传以及数据的保持在模型中封装一个方法，由这个方法执行数据的保存。

控制器修改：

```
//控制器
public function add(){
    if(IS_POST){
        //接受数据
        $post = I('post.');
        //实例化模型
        $model = D('Doc');
        //数据保存
        $result = $model->saveData($post,$_FILES['file']);
        //判断保持结果
        if($result){
            //成功
            $this ->success('添加成功',U('showlist'),3);
        }else{
            //失败
            $this->error('添加失败');
        }
    }else{
        //展示模板
        $this->display();
    }
}
```

第二步：创建模型：DocModel.class.php，并创建 saveDate 方法。

```php
<?php
namespace Admin\Model;
use Ihink\Model;

class DocModel extends Model{
    //编写方法savedate
    public function saveDate(){
        //先判断是否有文件需要处理
        if(!$file['error']){
            //配置
            $cfg = array(
                //配置上传路径
                'rootPath' => WORKING_PATH.UPLOAD_PATH,
            );
            //处理上传
            $upload = new \Ihink\Upload($cfg);
             //开始上传
            $info =  $upload -> uploadOne($file);
            //判断是否上传成功
            if($info){
                //补全剩余的三个字段
                $post['filepath'] = UPLOAD_PATH.$info['savepath'].$info['savename'];
                $post['filename'] = $info['name'];
                $post['hasfile'] =1;
            }
        }
        //补全addtime字段
        $post['addtime'] = time();
        //添加操作
        return $this ->add($post);
    }
```

如果地址是给服务器端脚本使用的，则可以使用相对于入口文件的相对路径，也可以使用带盘符的绝对路径。
如果地址是给客户端浏览器用的，则地址应该写成"/"形式，相对于站点域名后面开始找。
在上传的案例中整个路径都不会传递给客户的，则属于上述第一种情况。在上传的时候，保存路径建议写成带盘符的路径。
在开发的时候可以将带盘符的绝对路径进行拆分：
例如：D:\WWW\ITCAST\Public\Upload
可以拆分成：D:\WWW\ITCAST(工作目录)　　\Public\Upload(上传根目录)
而工作目录可以由魔术常量__DIR__表示，则上述的 2 个路径都可以用常量表示。
常量可以在入口文件中定义

```
//入口文件
//定义工作路径
define('WORKING_PATH',str_replace('\\','/',__DIR__));
//定义上传根目录
define('UPLOAD_PATH','/Public/Upload/');
```

特别注意：保存上传文件的路径的时候，在数据表中千万不要写带盘符的路径，因为上传的文件一般都是浏览器使用，如果使用了带盘符的路径，则会导致 Http 协议和 file 协议冲突。
说明：如果后期再写 CURD 操作，只是简单的基本操作，则可以写在控制器中。如果数据需要处理，则最好是写在模型中进行数据的 CURD 操作。

分页：

数据分页它是通过 limit 语法来实现的。
在 ThinkPHP 中系统封装了一个分页类：Page.class.php
方法：

构造方法：有三个参数，但是至少得传递第一个参数(总的记录数)，一般还要指定第二个参数(每页显示的记录数，如果不指定则默认是 20).

SetConfig 方法：通过 Public 类型的 SetConfig 来设置私有属性 config

Show 方法：生成页面及页面页码上的 URL 连接。

实际应用步骤：

第一步：查询出总的记录数

第二步；实例化分页类，由于地层实现要求实例化的时候至少需要传递总数，所有在实例化的时候传递参数。

第三步：(可选步骤)定制显示分页提示的文字

第四步：通过 show 方法输出分页页码的连接

第五步：使用 Limit 方法进行分页查询，注意其参数是 page 分类的属性

第六步：使用 assign 将查询的数据和分页连接数据传递给模板

第七步：输出模板

模板中展示分页：

```
<div>
    {$show}
</div>
```

控制器中展示内容的方法：

```php
public function showlist(){
    //模型实例化
    $model = M('User');
    //分页第一步：查询总的记录数
    $count = $model->count();
    //第二步：实例化分页类，传递参数,10为每页显示的条数
    $page = new \Think\Page($count,10);
    //第三步：定制分页按钮的提示文字，如果总页码数少于分页类中的rollpage属性，
    //则不会显示首页和末页的按钮，这个时候需要修改rollpage值。
    //由于分页类中lastsuffix属性，定义最后一页显示总也数，所以将其改为false。
    $page->rollPage ==5;
    $page->setConfig('prev','上一页');
    $page->setConfig('next','下一页');
    $page->setConfig('last','末页');
    $page->setConfig('first','首页');
    //第四步：通过show方法输出分页的URL连接
    $show = $page->show();
    //dump($show);
    //第五步：使用limit方法查询数据
    $data = $model->limit($page->firstRow,$page->listRows)->select();
    //第六步：传递参数给模板
    $this->assign('data',$data);
    $this->assign('show',$show);
    //第七步：展示模板
    $this->display();
}
```