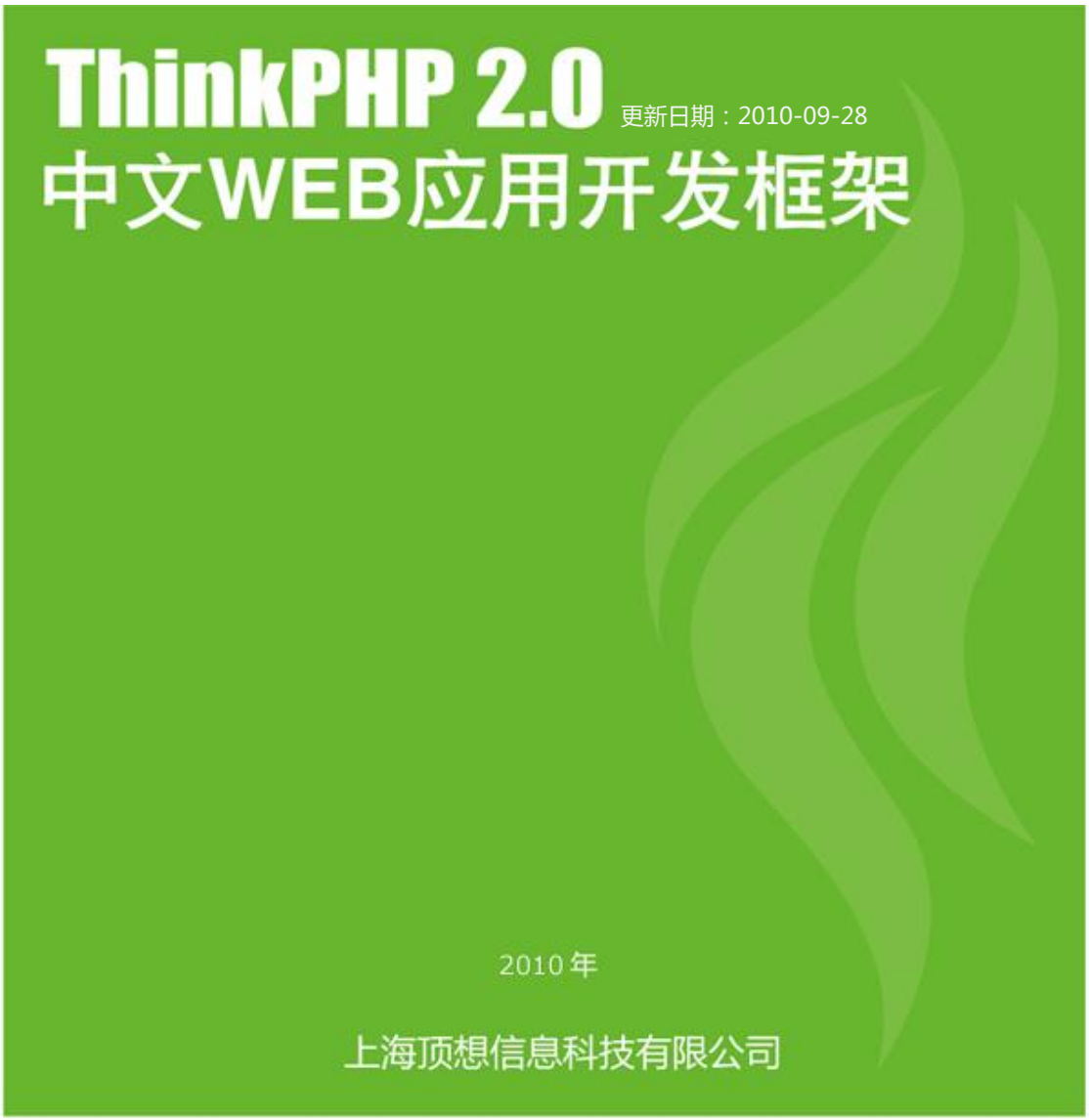
ThinkPHP
中文WEB应用开发框架

开发技巧



目 录

1 开发技巧 4

1.1 创建数据对象后的更改 4

1.2 定义实际的数据表名称 4

1.3 获取个别字段的值 5

1.4 设置字段别名 6

1.5 字段的表达式更新 7

1.6 字段的动态查询 7

1.7 针对主键的几个特殊用法 8

1.8 获取当前 Action 的名称 9

1.9 获取当前 Model 的名称 9

1.10 原生 SQL 和数据表替换 10

1.11 自动加载类库 11

1.12 文件哈希子目录缓存 13

1.13 使用正则表达式进行自动验证 13

1.14 不创建模型类如何自动验证 16

1.15 模型不需要数据库怎么定义 17

1.16 判断当前操作的请求类型 17

1.17 巧用空操作实现用户动态 URL 18

1.18 避免 URL 目录过深的技巧 19

1.19 利用路由实现用户动态 URL 20

1.20 巧用伪静态实现网站语言伪装 21

1.21 添加目录安全文件 22

1.22 避免 JS 代码被模板解析 23

1.23 模型单独设置数据表的前缀 24

1.24 巧用模型的表后缀实现多语言数据存储 25
1.25 空间不支持 PATHINFO 的处理 26
1.26 利用 ALLINONE 模式提高性能 27
1.27 设置默认时区 28
1.28 增加模板替换字符串 28
1.29 巧用公共文件检测浏览器缓存 30
1.30 使用 U 方法支持分组 30
1.31 定制页面 Trace 显示信息 31
1.32 支持 WML 32
1.33 利用初始化方法判断登录 32
1.34 巧用回调方法实现数组存储 33
1.35 定制 list 标签的字段列表 34
1.36 定制 list 标签的操作列表 36
1.37 主键不是 id 的时候 list 标签如何输出 37
2 推荐阅读 39
2.1 .htaccess 文件使用手册 39

1 开发技巧

1.1 创建数据对象后的更改

在使用 create 方法之后，我们仍然可以对创建的数据对象进行任何操作。

例如：

```
$User = M('User');
$User->create(); //创建 User 数据对象
$User->status = 1; // 设置默认的用户状态
$User->create_time = time(); // 设置用户的创建时间
$User->add(); // 把用户对象写入数据库
```

模型的 create 方法是用于创建数据对象，因为是创建到内存，因此在写入数据库之前可以随意添加或者更改。很多开发者为了需要自己设置字段，而放弃 create 方法，其实大可不必。

上面的处理方式有两个好处：

1、动态的更改字段可以避免在模型里面定义定义自动完成 或者直接使用 M 方法实例化模型而减少开销

2、可以解决在某些特殊的情况下难以统一定义自动完成的情况

1.2 定义实际的数据表名称

当你的数据表命名毫无规范可言（对于已有的数据库没有什么是不可能的~）没有统一规范的表前缀，表名大小写无规律，面对如此严峻的事实，你感到头脑发晕，但是千万不忘了 TP 的模型有一个终极武器，为每个模型定义实际的数据表名称。只需要在模型类里面添加

```
protected $trueTableName = '你的真实表名';
```

定义 trueTableName 属性后 会忽略当前的数据表前缀和 tableName 定义。而无论当前的模型名称是否和表名一致~

如果你的数据表还存在跨库的情况，可以加上数据库的定义

```
protected $dbName = '数据库名称';
```

1.3 获取个别字段的值

在连贯操作中 我们可以使用 field 方法来定义要返回的字段

```
$list = $User->field('id,name')->select();
dump($list);
```

会输出：

```
array(3) {
  [0] => array(2) {
    ["id"] => string(1) "1"
    ["name"] => string(5) "admin"
  }
  [1] => array(2) {
    ["id"] => string(1) "2"
    ["name"] => string(8) "thinkphp"
  }
  [2] => array(2) {
    ["id"] => string(1) "3"
    ["name"] => string(4) "test"
  }
```

如果不想返回数据集，而只是想返回一个以 id 为索引的包含 name 的数组，那么可以简单使用

```
$data = $User->getField('id,name');
dump($data);
```

会输出:

```
array(3) {
```

```
  [1] => string(5) "admin"
  [2] => string(8) "thinkphp"
  [3] => string(4) "test"
}
```

注意

```
$User->getField('name')和

$User->getField('id,name');
```

的返回值类型是完全不同的，前者只是返回 name 的值，并且始终只有一个。

输出的结果为：

```
string(5) "admin"
```

如果你只想取一个字段的值，但是希望返回数组的话，可以使用

```
$User->getField('id,id');
```

总结：模型类的 getField 方法是一个双关方法

1.4 设置字段别名

连贯操作的 field 方法可以用于设置查询的返回字段，根据数据库的查询优化建议，无论要返回多少字段，都尽量显示指定要查询的字段名。

今天我们要说的是如何在查询的时候指定字段别名，以及如何返回一些特殊的动态字段。

```
$User->field('id,nickname as name,status')->select();
```

这里把 nickname 设置成 name 别名后，查询结果里面就存在 name 字段而不存在 nickname 字段了。

利用这个技巧，我们可以实现一些实际并不存在的动态字段，例如，返回

```
$User->join('think_card card on think_user.id=card.user_id')->field('id,count(card.id) as card_count')->select();
```

1.5 字段的表达式更新

在使用 TP 的 save 方法或者 add 方法的时候，通常我们只能对数据对象赋简单的值，但是如果希望在字段写入的时候使用表达式的话，该如何处理呢？例如，我们希望完成下面这样的操作：

```
update think_user SET status=1,score= score+10 where name='thinkphp'
```

我们可以使用

```
$User->status =1;
$User->score = array('exp','score+10');
$User->where('name="thinkphp"')->save();
```

这个表达式的例子比较简单，其实可以使用更加复杂的，包括使用 mysql 的函数等等。

一旦使用 exp 的话，系统就会认为后面不再是一个值，而是一个表达式了。同样在 add 方法也可以使用字段表达式

1.6 字段的动态查询

ThinkPHP 提供了数据的动态查询方法，可以简化你的查询代码，例如：

```
$User->where('name="ThinkPHP"')->find();
```

可以简化为：

```
$User->getByName('ThinkPHP');
```

```
$User->where('email="thinkphp@qq.com"')->find();
```

可以简化为：

```
$User->getByEmail('thinkphp@qq.com');
```

getBy**** 方法里面的**** 会转换成小写的字段名，如果字段不存在，就会出错。

如果你的字段名是 user_id ，那么查询方法应该写成：

```
$User->getByUserId(5);
```

UserId 会被解析成为数据库的 user_id 字段，这点需要注意，以免引起不必要的麻烦。

目前尚不支持，对多个字段的动态查询。

1.7 针对主键的几个特殊用法

在 TP 的 CURD 中 有几个特别为主键的查询和删除方便而考虑的几个特殊用法：

1、**find**：

```
$User->find(3);
```

表示查询主键为 3 的用户记录

也可以支持字符串主键

```
$User->find('U4321');
```

需要注意的是，该方式的查询条件会覆盖 where 方法中定义的条件

```
$User->where('id=5')->find(3);
```

id=5 的查询条件将无效

2、**select**：

```
$User->select('1,2,5');
```

表示查询主键范围在 1，2，5 之内的用户数据

如果是字符串主键，需要注意

```
$User->select("'U1','U2','U5'");
```

每个主键要用引号

3、**delete**：

```
$User->delete(3);
```

表示删除主键为 3 的用户数据

```
$User->delete('3,5');
```

表示删除主键为 3 和 5 的用户数据

如果是字符串主键，需要写成

```
$User->delete("'U3','U5'");
```

1.8 获取当前 Action 的名称

由于某些原因，我们经常会在项目中定义一个公共的 Action，例如 CommonAction，然后在里面添加一些公共的操作方法，在这些公共方法里面，我们常常需要获取当前的 Action 名称，我们可以调用 Action 的 getActionName 方法，使用如下：

```
$name = $this->getActionName();
```

这样就能确保正确获取继承的 Action 类的 Action 名称。注意必须在 Action 类里面使用

早期 1.5 版本里面可以使用

```
$name = $this->name;
```

来完成同样的功能，但是由于和模板赋值机制有冲突，因此 2.0 版本开始就取消了，这点需要注意。

1.9 获取当前 Model 的名称

由于某些原因，我们经常会给项目定义一个公共的 Model，例如 CommonModel，然后在里面添加一些公共的查询方法，在这些公共方法里面，我们常常需要获取当前的 Model 名称，我们可以调用 Model 的 getModelName 方法，使用如下：

```
$name = $this->getModelName();
```

这样就能确保正确获取继承的 Model 类的 Model 名称。注意必须在 Model 类里面使用

早期 1.5 版本里面可以使用

```
$name = $this->name;
```

来完成同样的功能，但是由于和 ActiveRecord 特性的获取数据对象的属性有冲突，因此 2.0 版本开始就取消了，这点需要注意。

1.10 原生 SQL 和数据表替换

TP 的模型可以支持原生 SQL 操作，提供了 query 和 execute 两个方法，为什么原生 SQL 还要区分两个方法呢，原因有两个：

1、返回类型不同

query 用于查询，返回的是数据集，和 select 或者 findall 一样，所以可以直接在模板里面使用 volist 标签输出 query 的查询结果

execute 用于写操作，返回的是状态或者影响的记录数

2、读写统计需要

为了便于统计当前的数据读写次数，把数据库的读和写操作分开（对应的就是 query 和 execute）

使用原生 SQL 很简单，我们甚至不需要实例化任何的模型，例如：

```
$Model = new Model(); // 实例化一个空模型
```

下面的方法是等效的

```
$Model = D(); 或者 $Model = M();
```

```
// 下面执行原生 SQL 操作
$Model->query('select * from think_user where status=1');
$Model->execute('update think_user set status=1 where id=1');
```

如果你实例化了某个模型，仍然可以执行原生 SQL 操作，不受影响，例如：

```
$User = D('User');
$User->query('select * from think_user where status=1');
$User->execute('update think_user set status=1 where id=1');
```

在这种情况下面，我们可以简化 SQL 语句的写法，例如：

```
$User->query('select * from __TABLE__ where status=1');
$User->execute('update __TABLE__ set status=1 where id=1');
```

系统会自动把__TABLE__替换成当前模型对应的数据表名称，实际的数据表由模型决定。

通常来说，我们都是使用原生 SQL 操作实现一些 ORM 和 CURD 比较难实现的操作，另外，如果 SQL 不复杂的话 原生 SQL 的效率和连贯操作的效率差别是微乎其微的，TP 本身的 ORM 实现也是相当高效的。

1.11 自动加载类库

自动加载类库，是指在无需通过 require 和 TP 内置的 import 方法加载类库文件即可在需要的时候自动加载，自动加载机制可以让代码更简洁，并且利用得当的话，效率反而比手动加载有提升。

自动加载机制有四个方式：

1、列入系统的核心编译类别的类库都无需加载即可使用

2、定义了别名的类库会自动加载

3、当前项目的模型和 Action 类都会自动加载

4、自动搜索路径下面的类库可以自动加载

自动加载的类库文件命名必须是以 class.php 为后缀的。

从加载的效率上来看，依次递增，由此可见，给项目的类库定义别名是一个高效的方法，为项目中经常使用的类库定义别名是一个好的习惯，而且定义别名的副作用就是可以改变系统内置的自动定位机制。添加别名定义使用 alias_import 方法

```
alias_import(array(
    'myClass'  =>  LIB_ATH.'/Common/myClass.class.php',
    'myUtil'   =>  LIB_ATH.'/Common/myUtil.class.php',

    // ... 定义更多的别名

    )
);
```

我们可以在项目的公共文件 common.php 的最后添加这段代码。

然后，我们在其他地方直接使用类库，例如：

$class = new myClass();

这个时候系统会自动根据 myClass 定义的类库路径 自动加载找到 myClass 类。

如果你有很多的类库 不想一一定义别名的话，可以使用定义自动搜索路径的方法：

方法是定义 APP_AUTOLOAD_PATH 配置参数，该参数惯例配置的值是

'Think.Util.'，也就是说所有位于基类库 Think/Util/目录下面的类库都可以自动加载，但是我们还可以增加更多的搜索路径，例如：

```
'APP_AUTOLOAD_PATH'=>'Think.Util.,@.Common.',
```

这样的话，所有位于项目应用类库 Lib/Common/ 下面的类库也会自动加载。

1.12 文件哈希子目录缓存

ThinkPHP 内置的缓存可以支持包括 File、Db、Apc、Memcache、Shmop、Sqlite、Xcache、Apachenote、Eaccelerator 在内的缓存方式。系统的惯例配置默认使用文件方式缓存，也就是 File 方式。

文件方式默认的缓存目录位于项目的 Runtime/Temp/ 目录下面，并且不再区分子目录。如果你的缓存数据比较多，就可以启用哈希子目录缓存，只需要简单配置下面的参数：

```
'DATA_CACHE_SUBDIR'=>true, // 开启子目录缓存

'DATA_PATH_LEVEL'=>1 // 子目录层次 默认为 1
```

哈希子目录缓存仅对 File 方式的缓存有效。

设置以后，缓存文件会自动生成以缓存文件名的哈希规则为目录名的多层子目录。从而避免在同一目录下面缓存太多文件带来的性能损失。设置以后，缓存代码不用改变，还是使用：

```
S($name,$data); // 缓存数据

S($name);    // 获取缓存数据

S($name,NULL); // 删除缓存数据
```

1.13 使用正则表达式进行自动验证

TP 的自动验证机制是为了进行表单数据验证，验证可以支持 function、callback、confirm、equal、unique 和 regex，这里要讲的是使用正则表达式进行验证。

一般我们见的比较多的是设置规则为 require、email 之类的，其实这些本身也是属于正则表达式验证方式，只是系统内置定义了一些常用的正则表达式而已。这些内置的正则表达式的定义可以参考 model 类的 regex 方法，内置支持的正则定义包括：

require 字段必须、email 邮箱、url URL 地址、currency 货币、number 数字、zip 邮编、integer 整数、double 浮点数、english 英文字母，但是并不局限于这些正则规则的，我们完全可以直接在验证规则里面使用正则表达式进行定义，这样我们可以凭借强大的正则表达式来进行表单字段验证，例如：

```
array('name','/^[a-z]\w{3,}$/i','名字不符合要求！');

array('password','/^[a-z]\w{6,30}$/i','密码不符合要求！');

array('account','/^[A-Za-z]+$/','账号必须使用英文！');
```

附上一些表单验证中比较常用的正则表达式写法：

匹配中文字符的正则表达式： [\u4e00-\u9fa5]

匹配双字节字符(包括汉字在内)：[^\x00-\xff]

匹配 Email 地址的正则表达式：\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

匹配网址 URL 的正则表达式：[a-zA-z]+://[^\s]*

匹配帐号是否合法(字母开头，允许 5-16 字节，允许字母数字下划线)：^[a-zA-Z][a-zA-Z0-9_]{4,15}$

匹配国内电话号码：\d{3}-\d{8}|\d{4}-\d{7}

匹配中国邮政编码：[1-9]\d{5}(?!\d)

匹配 ip 地址：\d+\.\d+\.\d+\.\d+

匹配特定数字：

^[1-9]\d*$　　//匹配正整数

^-[1-9]\d*$　　//匹配负整数

^-?[1-9]\d*$　　//匹配整数

^[1-9]\d*|0$　　//匹配非负整数（正整数 + 0）

^-[1-9]\d*|0$　　//匹配非正整数（负整数 + 0）

^[1-9]\d*\.\d*|0\.\d*[1-9]\d*$　　//匹配正浮点数

^-([1-9]\d*\.\d*|0\.\d*[1-9]\d*)$　　//匹配负浮点数

^-?([1-9]\d*\.\d*|0\.\d*[1-9]\d*|0?\.0+|0)$　　//匹配浮点数

^[1-9]\d*\.\d*|0\.\d*[1-9]\d*|0?\.0+|0$　　//匹配非负浮点数（正浮点数 + 0）

^(-([1-9]\d*\.\d*|0\.\d*[1-9]\d*))|0?\.0+|0$　　//匹配非正浮点数（负浮点数 + 0）

匹配特定字符串：

^[A-Za-z]+$　　//匹配由 26 个英文字母组成的字符串

^[A-Z]+$　　//匹配由 26 个英文字母的大写组成的字符串

^[a-z]+$　　//匹配由 26 个英文字母的小写组成的字符串

^[A-Za-z0-9]+$　　//匹配由数字和 26 个英文字母组成的字符串

^\w+$　　//匹配由数字、26 个英文字母或者下划线组成的字符串

1.14 不创建模型类如何自动验证

我们知道，ThinkPHP 的模型有自动验证和自动完成功能，但是通常我们需要在模型类里面定义验证因子和完成因子。这样的话，我们使用 M 方法实例化模型的时候就不能使用内置的自动完成和自动验证功能了，其实仍然有办法的，因为 TP 提供了一个强大的属性动态更改的方法 setProperty。利用该方法就完全可以用 M 方法实现自动验证功能了，例如：

```
$User = M('User');
$auto = array (
array('status','1'), // 新增的时候把 status 字段设置为 1
array('password','md5',1,'function') // 对 password 字段在新增的时候使 md5 函数处理
array('name','getName',1,'callback') // 对 name 字段在新增的时候回调 getName 方法
array('create_time','time',2,'function'), // 对 create_time 字段在更新的时候写入当前时间戳
);
$validate = array(
array('verify','require','验证码必须！'), //默认情况下用正则进行验证
array('repassword','password','确认密码不正确',0,'confirm'), // 验证确认密码是否和密码一致
array('password','checkPwd','密码格式不正确',0,'function'), // 自定义函数验证密码格式
);
$User->setProperty('_auto',$auto);
$User->setProperty('_validate',$validate);
if($User->create()){
  $User->add();
}else{
  $this->error($User->getError());
}
```

完成自动验证和自动完成只是 setProperty 方法的一个小技巧而已，更强大的功能还需要你去发挥了。

1.15 模型不需要数据库怎么定义

TP2.0 开始，只要定义了模型就会连接数据库，1.5 版本有过虚拟模型的概念已经不存在了，那么，如果我们某些模型根本没有数据库操作，但是又想把一些业务逻辑封装到 model 类里面 怎么办呢？

其实，很简单，只要定义的 model 类不继承 Model 类即可，呵呵~例如：

```
class UserModel extends Think{
    // 添加自己的业务逻辑
}
```

类库命名还是保持 UserModel.class.php 不变，这样可以保证自动导入和 import 机制不变，另外，由于没有继承 Model 类，很多 Model 内置的方法和属性肯定不能再使用了。由于大多数方法都是和数据库操作相关的，所以也就无所谓了。

1.16 判断当前操作的请求类型

在很多情况下面，我们需要判断当前操作的请求类型是 GET POST 甚至是 PUT DELETE，一方面可以针对请求类型作出不同的逻辑处理，另外一方面有些情况下面需要验证安全性，过滤不安全的请求。

TP 的 Action 类内置了一些判断方法用于判断请求类型，包括：

isGet 是否是 GET 提交

isPost 是否是 POST 提交

isPut 是否是 PUT 提交

isDelete 是否是 DELETE 提交

isHead 是否是 HEAD 提交

使用举例如下：

```
class UserAction extends Action{
  public function update(){
    if ($this->isPost()){
      $User = M('User');
      $User->create();
      $User->save();

      $this->success('保存完成');

    }else{

       $this->error('非法请求');

    }
  }
}
```

另外还提供了一个判断当前是否属于 AJAX 提交的方法

isAjax 是否属于 AJAX 提交

需要注意的是，如果使用的是 ThinkAjax 或者自己写的 Ajax 类库的话，需要在表单里面添加一个隐藏域，告诉后台属于 ajax 方式提交，默认的隐藏域名称是 ajax（可以通过 VAR_AJAX_SUBMIT 配置），如果是 JQUERY 类库的话，则无需添加任何隐藏域即可自动判断。

1.17 巧用空操作实现用户动态 URL

在项目开发中，我们经常要实现类似 http://serverName/User/3 或者 http://serverName/User/张三这样的 URL

实现这样的功能有很多方法，这里说的是一种使用空操作的实现技巧。

要实现 http://serverName/User/3 这样的 URL，我们只需要在 UserAction 中定义空操作。

```
class UserAction extends Action{
public function _empty($id){
  // 空操作方法的参数其实就是 ACTION_NAME 这里就看成是用户编号
  $user = M('User')->find((int)$id);
  $this->assign('user',$user);
  // 在空操作中需要指定输出的模板 因为空操作并没有对应的模板
  $this->display('User:read');
}
}
```

同样的道理，我们可以实现 http://serverName/User/张三 这样的 URL

```
class UserAction extends Action{
public function _empty($name){
  $user = M('User')->getByName($name);
  $this->assign('user',$user);
  // 在空操作中需要指定输出的模板 因为空操作并没有对应的模板
  $this->display('User:read');
}
}
```

由于这只是一个示例，所以没有给出很健全的代码实现，请自行完善。

1.18 避免 URL 目录过深的技巧

按照 TP 的默认 URL 模式，通常是：

```
http://serverName/模块名/操作名/变量 1/值 1/变量 2/值 2...
```

很多人担心这样的 URL 会导致目录层次过深，而且由于这样的 URL 改变了当前的相对路径，所以如果不注意写法，经常会导致 JS 和 CSS 加载不到。问题就在于这个"/"，这两个问题都可以通过一个小技巧解决，而且不影响你的开发，只需要在项目配置文件中设置

```
'URL_PATHINFO_DEPR'=>'-',
```

这个配置默认值是"/" 我们更改为"-"

配置修改以后，上面的 URL 地址就可以变成：

```
http://serverName/模块名-操作名-变量 1-值 1-变量 2-值 2...
```

不过要注意的是，模板里面的链接地址最好是用 U 方法动态生成的，而不是固定写死的，否则模板会有一定的修改工作。

1.19 利用路由实现用户动态 URL

之前讲过利用空操作实现的用户动态 URL，其实路由也可以实现同样的功能，例如：

我们本来有一个 User 模块的 read 操作，现在的 URL 是

```
http://serverName/User/read/id/3
```

我们希望缩短成

```
http://serverName/User/3
```

现在无需修改任何代码，只需要增加路由定义即可。

要使用路由，确保首先开启路由

```
'URL_ROUTER_ON'=>true
```

然后在项目配置目录下面增加 routes.php 路由定义文件，添加下面的定义：

```
return array(
  'User@'=>array(
```

```
            array('/^\/(\d+)$/','User','read','id'),
        ),
    );
```

现在就可以使用

```
    http://serverName/User/3
```

和

```
    http://serverName/User/read/id/3
```

同时访问了

1.20 巧用伪静态实现网站语言伪装

TP 里面有一个 URL 伪静态的功能，本来是用于把 URL 伪装成一个静态页面地址用于 SEO 优化，例如设置：

```
    'URL_HTML_SUFFIX'=>'.html'
```

就可以实现

```
    http://serverName/Down.html
```

这样的 URL，实际上就是和

```
    http://serverName/Down
```

具有同样的作用。

稍微改进下，其实这个功能还可以起到表面上伪装网站技术的作用，呵呵~

例如，想忽悠客户你什么语言都能开发，遇到需要.Net 的客户我们改成

```
    'URL_HTML_SUFFIX'=>'.aspx'
```

就可以把

```
    http://serverName/Down
```

伪装成

```
http://serverName/Down.aspx
```

遇到要求 Java 的客户，我们可以改成

```
'URL_HTML_SUFFIX'=>'.do'
```

或者

```
'URL_HTML_SUFFIX'=>'.jsp'
```

遇到不专业的客户，没准还能被你忽悠过去了，哈哈~

1.21 添加目录安全文件

在有些服务器环境下面，是开启了 apache 的目录浏览权限的，这样就会导致用户可以通过 URL 访问到你的应用目录，查看到你有哪些模块和模板文件，显然对系统的安全性方面造成了一定的影响。

对于这样的情况，TP 提供了一个目录安全文件写入的功能，能够在项目的编译过程自动生成各个目录的安全文件，避免直接访问目录。要开启这个功能，我们只需要在项目的入口文件里面添加下面的定义：

```
define('BUILD_DIR_SECURE',true);
```

然后访问项目（必须在自动生成项目目录之前访问），这样就会自动给项目目录生成目录安全文件（默认会在相关的目录下面生成空白的 index.htm 文件），并且可以自定义安全文件的文件名 DIR_SECURE_FILENAME ，默认是 index.html，如果你想给你们的安全文件定义为 default.html 可以使用

```
define('DIR_SECURE_FILENAME', 'default.html');
```

还可以支持多个安全文件写入，例如你想同时写入 index.html 和 default.html 两个文件，以满足不同

的服务器部署环境，可以这样定义：

```
define('DIR_SECURE_FILENAME', 'index.html,default.html');
```

默认的安全文件只是写入一个空白字符串，如果需要写入其他内容，可以通过

DIR_SECURE_CONTENT 参数来指定，例如：

```
define('DIR_SECURE_CONTENT', 'deney Access!');
```

注意：

1、如果在后期设置，需要删除 Runtime 目录 才能重新生成目录安全文件

2、确保相关目录的可写权限

1.22 避免 JS 代码被模板解析

如果使用 TP 内置的模板引擎，而且采用默认的标签设置的话，在某些情况下，如果不注意，{$('name').value} 这样的标签很容易被内置模板引擎误解析。

解决这样的问题有三个方法，现列举如下：

1、{$('name').value}改成{ $('name').value}

因为内置模板引擎的解析规则是"{"后面紧跟"$"符号才会解析变量 因此只要在"{" 和"$"之间添加空格就不会被误解析了

2、使用内置的 literal 标签包含 JS 代码

<literal>JS 代码</literal> 包含在 literal 标签中的代码将会直接输出，不进行任何解析

3、定制模板引擎标签的定界符

例如：

```
'TMPL_L_DELIM'=>'<{',
'TMPL_R_DELIM'=>'}>'
```

1.23 模型单独设置数据表的前缀

我们知道，TP 的数据表前缀一般是统一定义的，但是如果个别数据表的前缀不统一的话应该如何处理，例如，大多数表的前缀是

think_ 而有两个表的前缀是 top_，应该怎么处理，解决办法是在模型里面单独设置自己的表前缀，例如：

我们在项目配置文件里面设置

```
'DB_PREFIX'=>'think_'
```

其实 TP 的惯例配置默认就是上面的设置，呵呵~

然后在 模型里面单独设置个别的表前缀，代码如下：

```
class UserModel extends Model{
    protected $tablePrefix = 'top_';
}
```

实例化 UserModel 后，实际连接的数据表就变成了 top_user，而不再是 think_user

1.24 巧用模型的表后缀实现多语言数据存储

在网站开发的时候经常需要用到多语言的网站版本，这里我们介绍一种巧用模型的后缀来实现的多语言表的连接。

假如我们需要存储网站的三个语言版本的数据，包括：简体中文、繁体中文和英文。

假设分别对应了数据库的三个表（以新闻表为例）

think_new_cn 简体中文（默认语言）

think_new_tw 繁体中文

think_new_en 英文

并且假设三个表的结构是一致的，只是表名不同。

然后，我们创建 New 模型如下：（注意我们只是创建了一个模型，而不是创建三个模型）

```
class NewModel extends Model{
    // 默认是简体中文
    protected $tableSuffix = '_cn';
    public function changeLang($lang){
        $this->trueTableName ='';//重置 trueTableName 属性 避免表名缓存

        $this->tableSuffix = '_'.$lang; // 切换表的语言后缀
        return $this;
    }
}
```

```
$New = D('New');
$list = $New->select(); // 查询简体中文
```

```
$New->changeLang('tw')->select(); // 切换到繁体中文表

$New->changeLang('en')->select(); // 切换到英文表

$New->changeLang('cn'); // 切换回简体中文
```

多语言的实现机制还有其他方式，以后我们会陆续介绍~

1.25 空间不支持 PATHINFO 的处理

经常遇到的一个问题就是，在本地测试环境没有任何问题，但是部署到客户的正式环境后，发现不管输入什么 URL 地址，访问的永远都是首页（也就是默认模块的默认操作），这个时候，第一感觉就是要查看空间是否支持 PATHINFO。

由于开发工作基本完成，这个时候再去改变 URL 地址的话，模板工作量会比较大。例如，可能需要把所有的类似

```
http://serverName/index.php/User/add
```

这样的 URL 地址改成

```
http://serverName/index.php?m=User&a=add
```

如果你的模板里面没有使用 U 方法统一生成 URL 的话，这个工作量随着模板文件的多少会有成倍的增长。

在这样的情况下面，最安全的方式，其实是调整 URL 模式，只需要做两步操作即可：

1、在项目配置文件里面设置：'URL_MODEL'=>3

2、清空模板缓存目录（通常默认是项目的 Runtime/Cache/）

这样设置后，系统的模板文件无需做任何更改，例如原来的模板里面的连接地址是

__URL__/add 或者 __APP__/User/add 这样的话，系统生成的链接

会自动变成

```
http://serverName/index.php?s=/User/add
```

经过这样的更改之后，大部分主机环境，包括国外的主机均可支持。

如果主机空间支持 REWRITE 和.htaccess 文件，还可以进一步处理 URL

修改你的.htaccess 文件为：

```
<IfModule mod_rewrite.c>
RewriteEngine on
RewriteCond %{REQUEST_FILENAME} !-d
RewriteCond %{REQUEST_FILENAME} !-f
RewriteRule ^(.*)$ /index.php?s=$1 [QSA,PT,L]
</IfModule>
```

经过这一步的处理后，那么你的 URL 仍然可以变回：

```
http://serverName/index.php/User/add
```

或者是

```
http://serverName/User/add/
```

关于在 IIS 下面使用 Rewrite 可以参考：

Apache 与 IIS,URL 重写机制的不同及 REQUEST_URI

1.26 利用 ALLINONE 模式提高性能

ALLINONE 模式指的是把核心编译缓存和项目编译缓存合并到一个文件里面去，并且过滤掉一些运行模式不需要执行的代码，并且对于用户的自定义常量全部统一定义，不再进行额外的检测。

ALLINONE 模式一般是在开发调试完成之后，希望进一步提高系统的整体性能的时候开启。开启 ALLINONE 模式只需要在入口文件中添加定义：

```
define('RUNTIME_ALLINONE', true); // 开启 ALLINONE 运行模式
```

开启 ALLINONE 运行模式后需要清空系统原来的编译缓存文件，第一次运行的时候系统会自动生成一个~allinone.php 的缓存文件，第二次就会直接读取缓存文件而跳过一些不必要的初始化过程。~allinone.php 编译缓存文件不是简单的~runtime.php 和~app.php 的合并，剔除了一些运行模式过程中不需要的方法和代码。

注意：在 ALLINONE 模式下面，即使调试模式开启也是无效的。

系统不支持对 ALLINONE 运行模式的开发调试功能。因此，大多数情况用于生产部署环境。

1.27 设置默认时区

有些应用所在的服务器和访问的区域间隔较大，导致服务器时间不准确，我们可以通过设置默认时区的方法来处理。

我们只需要在项目配置文件中添加：

```
'DEFAULT_TIMEZONE'=>'Asia/Singapore' // 设置默认时区为新加坡
```

注意：某些环境可能不支持时区设置，例如 PHP5.1 以下版本

1.28 增加模板替换字符串

我们知道，TP 提供了模板替换字符串功能，该功能主要可以帮助实现：

1、方便模板的本地单独预览；

2、方便在模板在不同的环境目录下面动态输出；

这个机制可以使得模板文件的定义更加方便，默认的替换规则有：

../Public： 会被替换成当前项目的公共模板目录 通常是 /项目目录/Tpl/default/Public/

__PUBLIC__：会被替换成当前网站的公共目录 通常是 /Public/

__TMPL__： 会替换成项目的模板目录 通常是 /项目目录/Tpl/default/

__ROOT__： 会替换成当前网站的地址（不含域名）

__APP__： 会替换成当前项目的 URL 地址 （不含域名）

__URL__： 会替换成当前模块的 URL 地址（不含域名）

__ACTION__：会替换成当前操作的 URL 地址 （不含域名）

__SELF__： 会替换成当前的页面 URL

注意：这些特殊的字符串是严格区别大小写的。

现在的问题是，如何定制项目需要的替换规则，比如我想更改某个替换规则并增加新的规则。

其实，很简单，我们只需要在项目配置文件中配置 TMPL_PARSE_STRING 就可以完成。如果有相同的数组索引，就会更改系统的默认规则。例如：

```
'TMPL_PARSE_STRING' => array(
    '__PUBLIC__' => '/Common', // 更改默认的__PUBLIC__ 替换规则
    '__UPLOAD__' => '/Public/Uploads/', // 增加新的上传路径替换规则
)
```

1.29 巧用公共文件检测浏览器缓存

1.5 版本的 TP 有一个浏览器缓存功能，新版内置取消了这个功能，那么应该如何使用呢？

其实我们可以借助 TP 的插件机制实现该功能，但是我们这里说的是一种取巧的方法，只需要做两件事情：

1、下载浏览器检测行为插件

可以通过 SVN 的 trunk/Addons/Behavior/BrowserCheckBehavior.class.php 更新或者在后面下载

下载后放入项目的 Lib\Behavior （如果没有请自行创建）目录下面。

2、执行行为

在项目的公共函数文件最后增加行为调用代码

```
B('BrowserCheck');
```

记得删除项目编译缓存文件~app.php 之后，这个行为才会生效。

因为这个检测行为是在应用的初始化步骤进行的，所以我们巧用公共函数文件的加载同时就调用了浏览器缓存检测行为。

下载地址：http://bbs.thinkphp.cn/misc.php?action=attachcredit&aid=1600&formhash=6994d15f

1.30 使用 U 方法支持分组

U 方法可以用来动态的生成需要的 URL 地址，如果使用了分组模式，我们仍然可以使用 U 方法。

```
U('/Admin-User/add');
```

生成的 URL 表示 Admin 分组的 User 模块的 add 操作，注意 Admin-User 之间的“-” 是固定的

同样的道理，我们在 Action 类的 redirect 方法中也可以这样使用

```
$this->redirect('/Admin-User/add');
```

1.31 定制页面 Trace 显示信息

页面 Trace 功能是 ThinkPHP 的一个用于开发调试的辅助手段。可以实时显示当前页面的操作的请求信息、运行情况、SQL 执行、错误提示等，启用调试模式的话，页面 Trace 功能会默认开启（除非在项目的调试配置文件中关闭），并且系统默认的 Trace 信息包括：当前页面、请求方法、通信协议、请求时间、用户代理、会话 ID、运行情况、SQL 记录、错误记录和文件加载情况。

如果需要扩展自己的 Trace 信息，有下面几种方式：

第一种方式：在当前项目的配置目录下面定义 trace.php 文件，返回数组方式的定义，例如：

```
return array(
    '当前页面'=>$_SERVER['PHP_SELF'],
    '通信协议'=>$_SERVER['SERVER_PROTOCOL'],...
);
```

在显示页面 Trace 信息的时候会把这个部分定义的信息追加到系统默认的信息之后，这种方式通常用于 Trace 项目的公共信息。

第二种方式：在 Action 方法里面使用 trace 方法来增加 Trace 信息，该部分可以用于系统的开发阶段调试。例如：

```
$this>trace('执行时间',$runTime);
$this>trace('Name 的值',$name);
$this>trace('GET 变量',dump($_GET,false));
```

1.32 支持 WML

我们只需要做如下步骤就可以支持 WML 开发：

1、设置模板后缀为.wml

在项目配置里面增加配置参数 'TMPL_TEMPLATE_SUFFIX'=>'.wml'

并确保你的所有模板文件都以 wml 后缀保存即可。并注意模板 wml 文件的定义规范~

2、更改模板输出类型

TP 默认的模板输出类型是 HTML，也就是 text/html，所以我们需要修改为 WML 的输出类型

```
$this->display('','','text/vnd.wap.wml');
```

display 方法的第三个参数是设置输出类型的，这个地方根据你的项目实际情况来，这里我们就使用 text/vnd.wap.wml

PS：最新的开发版本 SVN 已经增加了新的配置参数 TMPL_CONTENT_TYPE 因此更加方便了 不需要修改 display 方法。

1.33 利用初始化方法判断登录

如果我们需要判断用户是否登录，是否需要在每个模块的方法之前加上检测呢？答案显然是不需要。

假设现在我们的 Action 都继承了一个公共的 CommonAction 类，那么我们只需要在 CommonAction 类里面添加一个初始化方法，如下：

```
public function _initialize(){}
```

这个_initialize 方法会在所有操作方法之前首先执行，因此我们可以利用这个机制加上用户是否登录的判断

```
public function _initialize(){
    if (!isset($_SESSION['userId'])){

        $this->assign('jumpUrl','/Public/login'); //设置提示后跳转页面


        $this->error('请登录！');

    }
}
```

记得，所有需要登录检测的模块必须要继承 CommonAction 类，例如

```
class UserAction extends CommonAction{}
.....
```

1.34 巧用回调方法实现数组存储

在很多情况下面，我们需要存储一些数组数据到某个字段，例如兴趣爱好、多选分类等多选数据，解决的方式有很多，例如转换成逗号分割的字符串以及序列化的方式，处理方式可以利用自动完成或者字段过滤功能对该字段进行额外的处理，这里说的是一种透明的数组实现。我们利用的是模型的回调方法，解决方案如下：（我们假设模型是 DataModel，数组字段是 test）

```
class DataModel extends Model{

    // 写入数据前回调方法

    protected function _before_insert(&$data,$options) {
        $data['test']   =   implode(',',$data['test']);
    }

    // 更新数据前回调方法

    protected function _before_update(&$data,$options) {
        $data['test']   =   implode(',',$data['test']);
    }
```

```
    // 读取数据后的回调方法
    protected function _after_find(&$data,$options) {
        $data['test']  =  explode(',',$data['test']);
    }
}
```

经过这样的回调处理后，数组数据会在写入数据库之前把数组自动转成字符串，然后在读取数据库后把字符串自动转换成数组。

这样一来，我们在前端可以直接对该字段进行数组操作，例如 select checkbox 等表单类型。

总结：模型的回调方法用处比较广泛，这只是一个很简单的使用。

1.35 定制 list 标签的字段列表

TP 的 Html 标签库中有一个比较常用的 list 标签，可以用于后台直接输出数据集，并且集成了选择、排序、链接和操作功能。

其显示界面大致如图所示：

编号	标题	描述	添加时间	记录数	状态	操作
6	框架扩展	框架第三方扩展和插件	09-09-20 22:01	1		禁用 编辑 删除
5	插件扩展	TP的第三方扩展和插件	09-01-08 12:24	5		恢复 编辑 删除
4	应用示例	基于TP的开源应用和项目	08-12-16 11:11	5		禁用 编辑 删除
3	其他资源	其他相关图片和视频资源	08-12-11 15:48	25		禁用 编辑 删除
2	文档教程	官方相关文档和教程下载	08-11-21 23:21	67		禁用 编辑 删除
1	核心框架	框架各个版本的下载	08-09-29 13:53	16		禁用 编辑 删除

其中 list 标签最常用的应该是 show 属性了，其用于设置要显示的字段。如上图所示的字段设置如下：

```
    show="id:编号|8%,title:标题:read,content:描述,create_time|toDate='y-m-d H#i':添加时间,count:记录数,status|getStatus:状态"
```

上面的设置不包括多选框和操作列，我们来分析下字段显示的定义规则

```
字段 1|函数[^字段 2|函数,...]:显示名|宽度（像素或者百分比）:JS 方法链接|字段 1[^字段 2,...]
```

1、字段名

字段名就是要显示的数据表的字段名称，支持函数（这里指的是 PHP 里面的内置或者自定义函数）过滤，例如：

```
class_id|getClassName
```

和模板变量的过滤一样，前面的字段值就是后面的函数的参数，支持传入更多的参数：

```
create_time|toDate='y-m-d H#i'
```

如果要传入的参数是变量，可以使用

```
name|getShowName=$var
name|getShowName=$vo['name']
```

如果要在一个单元格里面同时显示多个字段的值，可以使用“^”分割，例如：

```
id^name:编号
```

并且也支持对多个字段使用函数功能

```
id|getUser^name|getShowName
```

字段的值使用函数过滤后，可以直接输出 Html 代码，甚至包括图片。

2、显示名

显示名比较容易理解，就是表格的头部显示的文字信息，可以设置列的宽度，包括使用像素或者百分比，例如：

```
name|200px
```

```
name|25%
```

3、超链接

超链接有两种实现方式：

一是使用字段的函数功能 直接在后台生成 html 超链接代码输出 这个部分就不作详细说明了。

二是使用 JS 超链接功能，给字段值添加一个 JS 方法的链接

例如，显示名称的编辑链接

```
name:名称:edit
```

edit 就是一个 JS 方法，默认情况下，edit 方法的参数是主键的值，而不是当前字段的值，如果需要指定字段的值，可以使用

```
name:名称:edit|name
```

还可以支持给 JS 方法传递更多的参数

```
name:名称:edit|id^name
```

JS 方法传递的参数都是当前字段的值，如果不是的话，建议直接写到 JS 方法里面。

利用上面的设置功能，能够输出满足需要的字段列表显示。list 标签的字段显示设置就讲到这里，其他的标签我们留到以后分析。

1.36 定制 list 标签的操作列表

list 标签可以支持操作列，对行数据进行统一的操作，可以支持下面的操作定义。

1、字段名|JS 函数|显示信息

新版推荐的用法，可以适用于大多数操作的定义 例如 cate_id|listCategory|查看类别

2、JS 函数:显示信息

这是一种针对主键的操作定义例如 edit:编辑

3、字段名|PHP 函数=其他参数

PHP 函数可以输出操作信息，例如： status|showStatus=$user['id'] 其中 user 是 list 标签的 name 属性定义的

4、JS 函数 1|JS 函数 2:信息 1|信息 2

旧版针对 status 字段的一个特殊用法 例如： forbid|resume:禁用|恢复

该用法已经不推荐使用 将来会废弃 建议用方法 3 替代

1.37 主键不是 id 的时候 list 标签如何输出

list 标签输出数据集的时候 默认的主键名称是 id，如果你的数据表主键名不是 id 的话，需要指定 pk 属性，例如：

```
<html:list id="checkList" name="user" style="list" checkbox="true" action="true"
datasource="list" show="id:编号|10%,name:变量:edit,title:名称,type|getAttrType:类型,value:
默认值,status|getStatus:状态|8%" actionlist="status|showStatus=$user['id']" />
```

假设当前的数据表主键是 user_id，那么 list 标签需要改成：

```
<html:list id="checkList" pk="user_id" name="user" style="list" checkbox="true"
action="true" datasource="list" show="id:编号|10%,name:变量:edit,title:名
称,type|getAttrType:类型,value:默认值,status|getStatus:状态|8%"
actionlist="status|showStatus=$user['id']" />
```

2 推荐阅读

2.1 .htaccess 文件使用手册

- .htaccess 文件(或者"分布式配置文件"提供了针对目录改变配置的方法， 即，在一个特定的文档目录中放置一个包含一个或多个指令的文件，以作用于此目录及其所有子目录。作为用户，所能使用的命令受到限制。管理员可以通过 Apache 的 AllowOverride 指令来设置

- 子目录中的指令会覆盖更高级目录或者主服务器配置文件中的指令。

- .htaccess 必须以 ASCII 模式上传，最好将其权限设置为 644。

2.1.1 错误文档的定位

常用的客户端请求错误返回代码：

401 Authorization Required

403 Forbidden

404 Not Found

405 Method Not Allowed

408 Request Timed Out

411 Content Length Required

412 Precondition Failed

413 Request Entity Too Long

414 Request URI Too Long

415 Unsupported Media Type

常见的服务器错误返回代码：

500 Internal Server Error

用户可以利用.htaccess 指定自己事先制作好的错误提醒页面。一般情况下，人们可以专门设立一个目录，例如 errors 放置这些页面。然后再.htaccess 中，加入如下的指令：

```
ErrorDocument 404 /errors/notfound.html
ErrorDocument 500 /errors/internalerror.html
```

一条指令一行。上述第一条指令的意思是对于 404，也就是没有找到所需要的文档的时候得显示页面为/errors 目录下的 notfound.html 页面。不难看出语法格式为：

```
ErrorDocument 错误代码 /目录名/文件名.扩展名
```

如果所需要提示的信息很少的话，不必专门制作页面，直接在指令中使用 HTML 好了，例如下面这个例子：

```
ErrorDocument 401 "你没有权限访问该页面，请放弃！"
```

2.1.2 文档访问的密码保护

要利用.htaccess 对某个目录下的文档设定访问用户和对应的密码，首先要做的是生成一个.htpasswd 的文本文档，例如：

```
zheng:y4E7Ep8e7EYV
```

这里密码经过加密，用户可以自己找些工具将密码加密成.htaccess 支持的编码。该文档最好不要放在 www 目录下，建议放在 www 根目录文档之外，这样更为安全些。

有了授权用户文档，可以在.htaccess 中加入如下指令了：

```
AuthUserFile .htpasswd 的服务器目录

AuthGroupFile /dev/null （需要授权访问的目录）
AuthName EnterPassword
AuthType Basic （授权类型）
```

```
    require user wsabstract （允许访问的用户，如果希望表中所有用户都允许，可以使用

require valid-user )
```

注，括号部分为学习时候自己添加的注释

2.1.3 拒绝来自某个 IP 的访问

如果我不想某个政府部门访问到我的站点的内容，那可以通过.htaccess 中加入该部门的 IP 而将它们拒绝在外。

例如：

```
    order allow,deny
    deny from 210.10.56.32
    deny from 219.5.45.
    allow from all
```

第二行拒绝某个 IP，第三行拒绝某个 IP 段，也就是 219.5.45.0~219.2.45.255

想要拒绝所有人？用 deny from all 好了。不止用 IP，也可以用域名来设定。

2.1.4 保护.htaccess 文档

在使用.htaccess 来设置目录的密码保护时，它包含了密码文件的路径。从安全考虑，有必要把.htaccess 也保护起来，不让别人看到其中的内容。虽然可以用其他方式做到这点，比如文档的权限。不过，.htaccess 本身也能做到，只需加入如下的指令：

```
    order allow,deny
    deny from all
```

2.1.5 URL 转向

我们可能对网站进行重新规划，将文档进行了迁移，或者更改了目录。这时候，来自搜索引擎或者其他网站链接过来的访问就可能出错。这种情况下，可以通过如下指令来完成旧的 URL 自动转向到新的地址：

```
Redirect /旧目录/旧文档名 新文档的地址
```

或者整个目录的转向：

```
Redirect 旧目录 新目录
```

2.1.6 改变缺省的首页文件

一般情况下缺省的首页文件名有 default、index 等。不过，有些时候目录中没有缺省文件，而是某个特定的文件名，比如在 pmwiki 中是 pmwiki.php。这种情况下，要用户记住文件名来访问很麻烦。在.htaccess 中可以轻易的设置新的缺省文件名：

```
DirectoryIndex 新的缺省文件名
```

也可以列出多个，顺序表明它们之间的优先级别，例如：

```
DirectoryIndex filename.html index.cgi index.pl default.htm
```

2.1.7 防止盗链

如果不喜欢别人在他们的网页上连接自己的图片、文档的话，也可以通过 htaccess 的指令来做到。

所需要的指令如下：

```
RewriteEngine on
RewriteCond % !^$
RewriteCond % !^http://(www\.)?mydomain.com/.*$ [NC]
RewriteRule \.(gif|jpg)$ - [F]
```

如果觉得让别人的页面开个天窗不好看，那可以用一张图片来代替：

```
RewriteEngine on
RewriteCond % !^$
RewriteCond % !^http://(www\.)?mydomain.com/.*$ [NC]

RewriteRule \.(gif|jpg)$ http://www.mydomain.com/替代图片文件名 [R,L]
```

2.1.8 二级域名配置,可以使项目中的多个分组呈现为二级域名的形式

例如: http://domain.cc/index.php/Admin 变为 http://admin.domain.cc/index.php

1.先配置域名

以 apache 为例

配置虚拟主机 DocumentRoot 为项目目录,ServerName 域名

#主域名

```
<VirtualHost *:80>
  DocumentRoot  D:\xampp\htdocs\demo2
  ServerName domain.cc
</VirtualHost>
```

#子域名

```
<VirtualHost *:80>
  DocumentRoot  D:\xampp\htdocs\demo2
  ServerName admin.domain.cc
</VirtualHost>
```

2.配置 host

以 windows 为例

在 C:\WINDOWS\system32\drivers\etc\hosts 中添加

127.0.0.1 domain.cc

127.0.0.1 admin.domain.cc*2.配置 host*

以 windows 为例

在 C:\WINDOWS\system32\drivers\etc\hosts 中添加

127.0.0.1 domain.cc
127.0.0.1 admin.domain.cc

3.修改程序的配置文件 config.php 如下

```
'APP_GROUP_LIST'    => 'Home,Test,Admin',
'DEFAULT_GROUP'=>'Home',

'APP_SUB_DOMAIN_DEPLOY'=>1, //激活子域名

/*子域名配置

*格式如: '子域名'=>array('分组名/[模块名]','var1=a&var2=b');

*/
'APP_SUB_DOMAIN_RULES'=>array(
    'admin'=>array('Admin/'),
    'test'=>array('Test/'),
),
```

ThinkPHP 文档小组 2010-9-28

HTTP://ThinkPHP.CN | WEB 应用开发最佳实践框架

大道至简，开发由我 WE CAN DO IT , JUST THINK