ThinkPHP® 2.1

常见问题&开发技巧

JUST THINK IT

JUST THINK IT

JUST THINK IT

JUST THINK IT

JUST THINK IT

上海顶想信息科技有限公司

目 录

1 常见问题

1.1 ThinkPHP 是什么？

ThinkPHP 是基于 MVC 模式的面向对象的 PHP 开发框架，基于 Apache2 开源协议发布，属于轻量级的中文 PHP 开发框架，提供 WEB 应用开发的快速解决方案。ThinkPHP 不是博客系统，也不是 CMS 系统，但是可以开发出任何类似博客或者 CMS 系统的应用出来。

1.2 ThinkPHP 是免费的么？

ThinkPHP 基于 Apache2 开源协议(http://www.apache.org/licenses/LICENSE-2.0)发布，并且永久免费下载和使用，并且对商业友好。

1.3 ThinkPHP 有 SVN 地址么？

ThinkPHP 有在 Google 项目申请注册，SVN 地址：

完整版本 http://thinkphp.googlecode.com/svn/trunk

核心版本 http://thinkphp.googlecode.com/svn/trunk/ThinkPHP

1.4 ThinkPHP 支持的 PHP 版本是多少？

ThinkPHP 2.*版本需要 PHP5.0（建议 5.2.0 以上版本）版本的支持。

1.5 ThinkPHP 有什么特殊的环境要求？

ThinkPHP 对服务器和操作系统环境没有太多要求，经过我们的测试在 Apache、IIS，甚至在 Nignx 下面都可以运行。并且核心框架没有依赖任何 PHP 的其他模块，只有在应用的时候才需要根据自己的需求来考虑是否需要额外的环境要求。

1.6 ThinkPHP 和其他框架比较有什么特色？

ThinkPHP 的主要特色是官方团队花费五年的时间和积累和打造的为用户考虑的众多细节，包括架构、功能和使用方面的一系列特点，并且功能全面、独具创新、文档齐全，是 WEB 应用开发的最佳实践框架。不但融合了众多不同语言和领域的优秀框架的思想，也给出了自己独有的功能创新和用户开发体验。其类库导入、项目编译、视图模型、ORM 实现、动态查询、分布式和多数据库支持、静态缓存、配置文件、缓存机制、模型自动验证和自动完成、空模块和空操作、权限认证、URL 模式等功能较国内外的框架有明显的不同和创新，内置了独立开发的基于 XML 和标签库的 PHP 模板引擎作为更是同类框架的首创。另外的优势就是本地化的文档和社区优势，官方提供有完全开发手册和在线手册。

1.7 ThinkPHP 里面的 MVC 对应哪些？

A 在 ThinkPHP 里面，M 是指模型类 Model V 是指模板文件 C 主要是指 Action 控制器。之所以说是主要，是因为有一些额外的控制操作是在核心控制器 App 中处理方面的，严格来说，他们也属于 C 的范畴。

1.8 什么是 CURD？

A CRUD 是一个数据库技术中的缩写词，一般的项目开发的各种参数的基本功能都是 CURD。它代表创建（Create）、读取（Read）、更新（Update）和删除（Delete）操作。CRUD 定义了用于处理数据的基本原子操作。.

1.9 什么是单一入口？

A 传统的模式下面，当 WEB 服务器收到一个 http 请求时，会解析该请求以确定要访问哪一个文件。例如 http://www.xxx.com/news.php 的解析结果就是要求 web 服务器解析 news.php 文件，并返回结果给浏览器。而单一入口则是无论什么功能操作都请求同一个 index.php 文件，然后根据 url 参数进行了第二次解析，以确定要访问的文件和操作，而 index.php 通常被称为入口文件。注意，单一入口并不代表网站是唯一入口的。

1.10 什么是系统基类库？

基类库位于框架系统目录下面的 Lib 目录，这些类库除了框架运行所需要的核心类库，还包括网站和项目开发过程中经常会用到的常用工具类。目前主要包含 Think 核心类库、ORG 扩展类库、Com 扩展类库，其中 Think 核心基类库的作用是完成框架的通用性开发而必须的基础类和常用工具类等，包含有：

Think.Core 核心类库包

Think.Db 数据库类库包

Think.Util 系统工具类库包

Think.Template 内置模板引擎类库包

Think.Exception 异常处理类库包

ORG 和 Com 类库包主要用于基类库的扩展，ORG 类库包主要是第三方的公共类库，而 Com 类库包通常用于企业类库，或者多年的开发经验而积累形成的类库包，主要是内部或者局部范围使用。默认情况下，ORG 类库包是已经有创建的，并且也包含了一些常用的类库，而 Com 类库包是需要自己来创建的，因此你在系统的 Lib 目录下面是看不到 Com 目录的。

1.11 ThinkPHP 是低耦合的框架么？

可以这么认为，ThinkPHP 的核心（类库）是高耦合的，这是出于效率和机制的考虑，因为 ThinkPHP 的惯例配置贯穿始终，因此，ThinkPHP 的核心是不可拆分的（这里指的核心是指 Think 基类库），但是保留了扩展机制。而扩展类库是低耦合的或者可替换的。

1.12 ThinkPHP 可以使用第三方的类库或者类库包么？

A ThinkPHP 完全可以支持第三方的类库引入，放入 Vendor 目录后即可直接使用。导入方式参考下面的方式：Vendor('Zend.Filter.Dir'); 利用这个机制，我们完全可以把其他框架的低耦合类库直接在 ThinkPHP 中调用。

1.13 ThinkPHP 的类库一定要使用.class.php 后缀么？

A ThinkPHP 的默认规范是类库名和文件名一致（包括大小写），并且后缀使用.class.php，这是为了更加方便系统内置的类库导入（import）机制。如果你的后缀使用.php 的话，一样可以通过参数控制导入，并不会影响正常使用。

1.14 可以让编译缓存保留空白和注释或者关闭编译缓存么？

A 默认的情况下，为了缩小文件大小，系统对核心编译缓存和项目编译缓存文件去掉了空白和注释，但是可以通过如下的配置保留，以进行更加方便的调试定位。在入口文件里面添加下面的常量定义即可：

```
define('STRIP_RUNTIME_SPACE',false);
```

还可以关闭核心编译缓存（同样在入口文件里面定义）

```
defined('CACHE_RUNTIME',false);
```

如果开启了项目的调试模式的话，也会关闭项目的编译缓存。

1.15 ThinkPHP 是如何识别和解析 URL 的？

ThinkPHP 里面会根据当前的 URL 来分析要执行的模块和操作。这个分析工作由 URL 调度器来实现，官方内置了 Dispatcher 类来完成该调度。在 Dispatcher 调度器中，会根据

http://domainName/appName/moduleName/actionName/params

来获取当前需要执行的项目（appName）、模块（moduleName）和操作（actionName），在某些情况下，appName 可以不需要（通常是网站的首页，因为项目名称可以在入口文件中指定，这种情况下，appName 就会被入口文件替代），另外针对不同的 URL 模式设置系统会进行不同的智能识别。

1.16 ThinkPHP 如果获取带有/的 get 参数？

在默认的情况下，TP 采用的 PATHINFO URL 模式，并且默认分隔符是”/“，这个时候，如果传递带有“/”的参数就会发生混淆，处理方法是对传人的 get 参数进行处理，例如对参数进行 url_encode、**rawurlencode** 或者 base64_encode 编码处理，或者更改默认的 PATHINFO 分隔符。

1.17 出现缓存文件写入失败是什么原因？

检查下缓存目录是否具有可写权限，TP 默认情况下，要求 Runtime 目录及其子目录都具有可写权限。

1.18 ThinkPHP 里面必须给每个操作定义方法么？


对于没有任何业务逻辑的操作我们可以直接定义模板文件即可，这种情况下无需定义操作方法，系统会直接渲染模板文件输出。

1.19 ThinkPHP 可以支持多主题吗？


ThinkPHP 支持主题的概念，默认的是 default 主题，如果需要增加新的主题，只需要在项目目录下面的 Tpl 目录下面创建新的主题目录即可，然后在项目配置中设置默认的主题名称：

'DEFAULT_THEME' => 'new'

也可以在程序中动态改变当前的模板主题，例如根据不同用户设置不同的皮肤模板风格等等。

1.20 ThinkPHP 的 display 方法是如何定位模板文件的？


ThinkPHP 的模板输出通常只需要写一个空白的 display 方法

```
$this->display();
```

该方法没有定义任何要输出的模板文件，但是系统会根据默认的规则去寻找模板目录下面的以模块目录的操作模板文件。例如，假如 display 方法位于 UserAction 类的 add 操作方法，那么系统会自动寻找模板目录下面的 User/add.html 模板文件，这就是为什么空的 display 方法也能输出模板的原因。当然，display 方法一样可以支持参数输出，而且有很多的规则。

1.21 ThinkPHP 的控制器名称是否一定要和数据表一致？

A ThinkPHP 的控制器（Action）类和数据表完全没有关系，怎么命名取决于你的项目如何进行模块化的设计。

1.22 如何让网站默认访问 Blog 模块而不是 Index 模块

A 可以在项目配置文件里面配置：

'DEFAULT_MODULE'=>'Blog'

就可以改变默认的模块访问。

1.23 入口文件里面的 THINK_PATH 应该如何定义？

A 入口文件里面的 THINK_PATH 主要用于定位 ThinkPHP 系统目录的位置，可以使用相对路径或者决定路径都可以，例如：

define('THINK_PATH', '../ThinkPHP');

或者

define('THINK_PATH', '/Home/ThinkPHP');

如果该位置已经加入 PHP 的搜索路径，可以无需定义。也就是说，框架的系统目录可以随意放置，哪怕是不在 WEB 访问路径下面。

或者干脆不用定义 THINK_PATH，而是直接包含框架的系统入口文件，例如把原来的：

define('THINK_PATH', '../ThinkPHP');

require(THINK_PATH."/ThinkPHP.php");

改成：

require ("../ThinkPHP/ThinkPHP.php");

1.24 入口文件中的项目路径应该如何定义？

入口文件里面的项目路径指的是项目目录（也就是项目 Lib、Conf 所在的目录）所在的路径，和项目的入口文件的位置没有关系，原则上，项目的入口文件可以随意摆放，只要是在 WEB 访问目录下面即可。只是随着项目入口文件的位置不同，项目路径的定义也会变化。项目路径的定义和 THINK_PATH 定义一样，采用相对路径和绝对路径都可以。

1.25 ThinkPHP 是否支持 PATHINFO？

ThinkPHP 可以完美支持 PATHINFO，并且可以配置 PATHINFO 方式的 URL 分隔符，例如可以支持下面的 URL

http://domainName/User/read/id/1

http://domainName/User-read-id-1

1.26 ThinkPHP 必须要求服务器支持 PAHTINFO 么？

ThinkPHP 除了 PATHINFO 模式外，还可以支持普通 URL 模式和兼容 URL 模式，这两种模式都可以用于不支持 PATHINFO 的服务器环境。例如，原来的 URL 可能变化为：

普通 URL 模式 ：http://domainName/?m=User&a=read&id=1

兼容 URL 模式 ：http://domainName/?s=/User/read/id/1

例如对于 Nginx 环境和个别 FASTCGI 模式（国外的空间居多）下面有可能不支持 PATHINFO 模式，官方建议采用兼容 URL 模式进行处理，这样的方便之处是可以和 PATHINFO 模式实现配置切换，而不需更改任何模板文件。

1.27 ThinkPHP 怎么实现动态的 URL 解析？

例如 http://domainName/User/3 这样的 URL 地址，因为 User 后面的 id 是一个可变的参数，所以无法当成一个普通的操作名称来解析，ThinkPHP 里面有多种方案可以实现类似的 URL，包括：

使用 URL 路由功能把 User/3 路由到 User 模块的 read 操作

使用空模块和空操作功能

第二种方案还可以实现 http://domainName/3 这样的 URL

1.28 ThinkPHP 是否支持路由？

ThinkPHP 可以很好的支持路由功能，包括简单路由和正则路由支持，可以通过配置来支持实现不同的功能，例如：

http://domainName/Blog/3

http://domainName/Blog/2008/12/

1.29 ThinkPHP 是否支持 SEO 优化？

A ThinkPHP 可以针对 URL 进行定制 在一定程度上可以满足 SEO 对 URL 设计的要求。例如，可以实现类似下面的 URL 地址：

http://domainName/blog/3

http://domainName/blog-3.html

http://domainName/blog_2008_12

1.30 ThinkPHP 的验证码为何无法显示？

A 验证码无法正常显示，通常包括几个原因：

检查你的 GD 库模块是否开启；

检查你的程序在验证码之前是否有任何输出；

如果使用了 UTF8 编码，请确保删除 UTF8 的 BOM 信息头；

1.31 ThinkPHP 可以同时执行多个操作么？

A ThinkPHP 的操作链功能可以按照顺序执行多个定义好的操作，其访问格式是：

http://domainName/appName/User/action1:action2:action3/

上面的访问地址可以同时执行 User 模块的 action1、acton2 和 action3 操作方法。

1.32 ThinkPHP 可以支持哪些数据库？

A ThinkPHP 可以支持的数据库包括 Mysql、MsSQL、PgSQL、Sqlite、Oracle、Ibase，更多的数据库支持还可以使用 PDO 方式连接。

1.33 ThinkPHP 是否会自动关闭数据库连接？

A ThinkPHP 在完成数据操作后，会自动关闭数据库连接，这个操作是在 Db 类的析构方法里面完成的，如果你需要更加及时的关闭数据库连接。

1.34 ThinkPHP 的模型类的名称必须要和数据表一致么？

A ThinkPHP 的模型（Model）类和数据表可以不一致，你只要设置模型类的 tableName 或者 trueTableName 属性即可。默认情况下保持和数据表一致的定义是为了让系统可以自动识别对应的数据表。系统可以自动识别的模型名称定义包括下面两种方式(假设数据表的前缀定义是 think_)：

模型名和数据表名一致（不包括数据表的前缀和后缀），例如：UserModel 可以字段对应数据表 think_user

模型名采用驼峰法命名，例如：UserTypeModel 可以自动对应数据表 think_user_type

1.35 ThinkPHP 是否支持跨数据库和跨服务器操作？

ThinkPHP 的模型类可以定义单独的数据库名称，查询的时候会自动加上当前所属的数据库。还可以给模型定义单独的数据库连接，可以使得某个模型可以支持不同的服务器上面的不同数据库类型。要注意的是跨服务器的查询要避免使用视图和 JOIN 查询。

1.36 为什么修改了数据表的字段后无法识别新的字段？

ThinkPHP 会对数据表的字段信息进行缓存，如果你在开发过程中修改了数据表的字段而又没有开启调试模式的话，会发生对应的字段无法保存的情况，这个时候需要删除 Runtime/Data/_fields 目录下面的字段缓存文件，在调试模式下面会自动关闭数据表字段缓存。

你也可以自己在项目配置里面添加：

```
'DB_FIELDS_CACHE'=>false,      // 不缓存数据表的字段信息
```

1.37 可以在模型类里面定义数据表的字段信息吗？

答案是肯定的，如果不希望 ThinkPHP 自动去获取数据表的字段信息并缓存，可以直接在模型类里面定义好相关的字段信息，这样的好处是可以节省文件读取的 IO 开销，建议在部署的时候可以使用，缺点是每次更改数据库的字段都必须手动更新。手动定义的格式是在模型类添加下面的格式定义：

```
protected $fields = array(
    'id', 'username','email',          'age',     // 字段信息
```

```
    '_pk'=>'id',                // 主键名称
    '_autoInc'=>true        // 主键是否属于自动增长类型
)
```

1.38 ThinkPHP 的表单字段是否一定要和数据表的字段保持一致？

默认情况下，表单提交的字段名要和数据表的字段保持一致，否则无法写入到数据表里面。但是，系统提供了表单字段映射功能，可以给数据表的字段定义表单映射，来避免用户直接知道数据表的字段设计。

1.39 ThinkPHP 是否支持分布式数据库？

ThinkPHP 内置支持分布式数据库的定义和查询，包括读写分离。可以参考如下的项目配置：

在项目配置文件里面定义

```
Return array(
'DB_DEPLOY_TYPE '=>1, // 启用分布式数据库支持
'DB_RW_SEPARATE'=>true,  // 设置读写操作分离
'DB_TYPE'=> 'mysql', // 分布式数据库的类型必须相同
'DB_HOST'=> '192.168.0.1,192.168.0.2',    // 分布式数据库的地址
'DB_NAME'=>'thinkphp', // 如果相同可以不用定义多个
```

```
'DB_USER'=>'user1,user2',
'DB_PWD'=>'pwd1,pwd2',
'DB_PORT'=>'3306',
'DB_PREFIX'=>'think_',
..... 其它项目配置参数
);
```

但是数据库的数据同步不是由框架自动完成，而是交给数据库本身来实现。

1.40 ThinkPHP 是否支持同时多个数据库连接？

在系统的配置数据库连接之外，ThinkPHP 可以良好的支持多个数据库的连接和切换。这个连接是动态的，在程序里面实现。例如：

```
$User = D("User");
// 创建多个数据库连接的 DSN
$myConnect1 = 'mysql://username:passwd@192.168.1.1/DbName1';
$myConnect2 = 'pgsql://username:passwd@192.168.1.2/DbName2';
// 增加数据库连接 第二个参数表示连接的序号
// 注意内置的数据库连接序号是 0,所以额外的数据库连接序号应该从 1 开始
// 可以同时增加多个数据库连接
$User->addConnect($myConnect1,1);
```

```
$User->addConnect($myConnect2,2);
// 切换当前要操作的数据库到连接 2
$User->switchConnect(2);
// 关闭连接序号为 2 的数据库连接
$User->closeConnect(2);
```

1.41 模型的自动验证和自动完成有什么区别？

A 模型的自动验证主要是对表单提交的数据进行验证是否符号要求，自动完成是对表单没有的数据进行添加，或者对提交的数据进行过滤，两者的配合可以保证写入数据库的数据是符合要求的数据信息。这两个功能需要数据的创建是使用 Create 方法创建的，如果没有使用 Create 方法创建数据对象的话，可以使用高级模型提供的字段自动过滤功能。

1.42 自动验证的 Callback 和 function 方式定义的区别是什么？

A callback 表示是模型的方法调用， function 表示是函数调用

1.43 ThinkPHP 的视图模型是什么含义？

A ThinkPHP 在 ORM 模型里面模拟实现了数据库的视图模型，该功能可以用于多表查询。和数据库的视图概念区别是视图模型修改比较方便，无需修改数据库本身。而且不需要数据库的视图支持，

这是 ThinkPHP 框架的亮点之一。下面定义了一个 Blog 视图模型，我们通过创建 BlogView 模型来快速读取一个包含了 User 名称和类别名称的 Blog 记录（集），其查询方式和普通模型一样。

```
class BlogViewModel extends Model
{
    protected $viewModel = true;
    protected $viewFields = array(
        'Category'=>array('title'=>'categoryName'),
        'User'=>array('name'=>'userName'),
        'Blog'=>array('id','name','title'),
        );
}
```

1.44 ThinkPHP 的配置文件采用什么格式？

A ThinkPHP 的配置文件采用效率最高的 PHP 返回数组方式，不需要额外的解析过程。只要会使用 PHP 的数组，就会定义 ThinkPHP 的配置文件。

1.45 ThinkPHP 的配置文件的优先级别是什么？

A ThinkPHP 中配置文件的优先顺序从低到高依次是：（在没有生效的前提下）

惯例配置 → 项目配置 → 调试配置 → 模块配置 → 操作（动态）配置

1.46 ThinkPHP 的惯例配置有哪些参数？

A ThinkPHP 的系统惯例配置文件在系统目录的 Common\convention.php，里面列出了系统的所有配置参数以及默认配置，并且有详细的注释，具体也可以参考官方的配置指南和参考文档。

1.47 ThinkPHP 的模块配置的名称格式是什么？

A ThinkPHP 的模块配置文件位于项目的 Conf 目录下面，命名规范是：

模块名称（小写）_config.php

1.48 ThinkPHP 的命名规范有哪些？

A ThinkPHP 的开发过程中尽量遵循下面的文件命名规范：

类文件都是以.class.php 为后缀（这里是指的 ThinkPHP 内部使用的类库文件，不代表外部加载的类库文件），使用驼峰法命名，并且首字母大写，例如 DbMysql.class.php。

函数、配置文件等其他类库文件之外的一般是以.php 为后缀（第三方引入的不做要求）。

确保文件的命名和调用大小写一致，是由于在类 Unix 系统上面，对大小写是敏感的（而 ThinkPHP 在调试模式下面，即使在 Windows 平台也会严格检查大小写）。

类名和文件名一致（包括上面说的大小写一致），例如 UserAction 类的文件命名是 UserAction.class.php， InfoModel 类的文件名是 InfoModel.class.php，

函数的命名使用小写字母和下划线的方式，例如 get_client_ip

Action 控制器类以 Action 为后缀，例如 UserAction、InfoAction

模型类以 Model 为后缀，例如 UserModel、InfoModel

方法的命名使用驼峰法，并且首字母小写，例如 getUserName

属性的命名使用驼峰法，并且首字母小写，例如 tableName

以双下划线“__”打头的函数或方法作为魔法方法，例如 __call 和 __autoload

常量以大写字母和下划线命名，例如 HAS_ONE 和 MANY_TO_MANY

配置参数以大写字母和下划线命名，例如 HTML_CACHE_ON

语言变量以大写字母和下划线命名，例如 MY_LANG，以下划线打头的语言变量通常用于系统语言变量，例如 _CLASS_NOT_EXIST_。

数据表和字段采用小写加下划线方式命名，例如 think_user 和 user_name

1.49 如何关闭 ThinkPHP 的模板缓存？


ThinkPHP 的模板缓存是无法关闭的，因为内置的模板引擎是一个编译型的模板引擎，必须经过编译后生成一个可执行的缓存文件才能被执行。但是可以设置缓存的有效期，例如设置

```
'TMPL_CACHE_TIME'        =>3,        // 模板缓存有效期 -1 永久 单位为秒
```

这样，每隔 3 秒系统会自动重新编译模板文件。默认的配置是-1 表示永久缓存，除非模板文件有改动，模板文件一旦有改动会自动重新编译，如果是包含进来的外部文件有修改，系统是不会自动重新编译的。

1.50 ThinkPHP 的模板如何使用 PHP 本身作为模板引擎？


ThinkPHP 内置的模板引擎也支持直接在模板文件里面使用 PHP 代码，如果你不想使用任何模板引擎和标签的话，可以配置模板引擎类型为 PHP 就可以完全使用 php 本身作为框架的模板引擎，在项目配置里面添加：

'TMPL_ENGINE_TYPE' =>'php'

1.51 ThinkPHP 的模板可以使用第三方的模板引擎吗？


ThinkPHP 框架允许你使用第三方的模版引擎。目前官方已经提供了 Smarty 模版引擎的插件，已经有人给 ThinkPHP 开发了 TemplateLite、EaseTempalte 和 DzTemplate 模版引擎插件。而且对于自己熟悉的模版引擎来说，非常容易扩展类似的插件。然后在项目配置文件里面配置使用何种模板引擎就可以了。

1.52 如何输出其他模块的操作模板？


系统提供的 display 方法支持调用不同位置的模板文件，包括其他模块的操作，例如下面的方法可以调用 Member 模块的 read 操作模板：

```
$this->display('Member:read');
```

1.53 模板文件开头的<taglib name=’html’ />是什么意思？


这表示当前模板文件要加载 html 标签库，这样在模板文件里面就能使用类似

<html:list > <html:link > 之类的标签了，内置的模板引擎是基于标签库和 XML 解析的，所以必须要引入相应的标签库才能进行标签解析，因为系统默认会加载 cx 标签库，所以

<volist > <eq > 这样的标签是不需要自己加载标签库的。Cx 标签库之外的都需要在模板文件的开头用<tagLib 标签首先引入标签库。

1.54 编辑器无法识别 XML 标签，模板标签的定界符可以定制吗？


内置的模板引擎默认采用的是 XML 标签作为标签的定界符，但是可以修改的，下面是系统默认的配置，包括普通模板引擎和标签库的标签的起始和结束标记：

```
'TMPL_L_DELIM'=>'{',            // 模板引擎普通标签开始标记
'TMPL_R_DELIM'=>'}',            // 模板引擎普通标签结束标记
'TAGLIB_BEGIN'=>'<',                    // 标签库标签开始标记
```

'TAGLIB_END'=>'>',　　// 标签库标签结束标记

需要注意的两种类型的标记不要设置为相同的，以免引起混淆而无法正常解析。

1.55　如何获取模板输出的内容？

A Action 类的 display 方法是用于渲染模板文件并输出，可以使用 fetch 方法渲染模板文件但不是直接输出，而是返回内容。

1.56　模板文件里面经常使用的__URL__和 __APP__有什么用？

A 如果使用了内置模板引擎的话，可以在模板文件里面使用一些已经定义好的特殊字符串，系统在输出模板的时候会自动替换成相关的系统常量，这些可替换的字符串包括：

../Public　//项目公共目录

__PUBLIC__　　//网站公共目录

__ROOT__　　//网站根目录

__TMPL__　　//当前模板目录

__APP__　//当前项目地址

__URL__　//当前模块地址

__ACTION__　　//当前操作地址

__SELF__　//当前页面地址

1.57 如何在模板文件中直接输出系统变量和常量？


系统变量，必须以$Think.打头，如

{$Think.server.script_name } //取得$_SERVER 变量

{$Think.session.session_id } // 获取$_SESSION 变量

{$Think.get.pageNumber } //获取$_GET 变量

{$Think.cookie.name } //获取$_COOKIE 变量

输出系统常量

{$Think.const.__FILE__ }

{$Think.const.MODULE_NAME }

1.58 ThinkPHP 调试起来方便吗？


ThinkPHP 支持调试模式，在调试模式下面系统默认开启了日志记录、关闭了字段缓存、关闭了模板缓存，记录了执行过程中的 SQL 语句和运行时间，并且开启了页面运行时间显示和 Trace 功能。

要开启调整模式，在项目配置中添加如下设置：

'APP_DEBUG' => true

在程序的数据调试输出方面，我们提供了非常有用的方法：

dump($var) //在浏览器输出方便查看的变量信息

halt($msg) //输出信息，并中止执行

1.59 ThinkPHP 的页面 Trace 是什么作用？


页面 Trace 功能是 ThinkPHP 用于调试当前页面状态信息、错误记录和 SQL 记录，是一个非常有帮助的调试手段。而且开发人员可以定制需要显示的信息。

1.60 怎么查看上次执行的查询语句是什么？


如果你开启了调试模式，可以在日志文件里面查看最近执行过的 sql 语句和执行时间。如果没有开启，也可以使用调试方法来查看，下面的代码可以查看上次执行的 SQL 语句：

```
$User = D('User');
// 执行查询
$User->where('status>1')->order('create_time desc')->limit(10)->findAll();
// 查看上次执行的 SQL 语句
echo $User->getLastSql();
```

1.61 经常看到的 D 方法和 C 方法是什么意思？

ThinkPHP 为一些常用的操作定义了快捷方法，这些方法具有单字母的方法名，具有比较容易记忆的特点，D 方法和 C 方法是其中用的比较多的。

D 方法用于快速创建模型对象的实例，并且单例化，例如：

$User = D("User");

等效为

$User = new UserModel();

C 方法用于快速获取和修改配置参数，例如：

设置名称为 USER_AUTH_ON 的配置参数

C('USER_AUTH_ON',true);

获取 USER_AUTH_ON 的变量值

C('USER_AUTH_ON');

除了 D 和 C 方法外，系统还提供了 A、S 和 L 方法，具体可以查阅手册。

1.62 ThinkPHP 是否可以支持 Jquery 和其他的 JS 框架？

ThinkPHP 框架本身不会干涉客户端的任何东西，包括 JS，唯一一个的区别是因为框架采用了单一入口和默认的 PATHINFO 模式，改变了传统的 URL 路径，也就是说所有所有的调用路径应该都是基于入口文件的 URL 位置来。

1.63 如何采用子目录的方式缓存数据？

A 系统默认的文件缓存是在同一个目录下面的，也就是 Runtime/Temp 目录，但是可以配置启用子目录缓存：

'DATA_CACHE_SUBDIR'=>True

这样，系统会自动生成文件的哈希子目录来存放数据缓存，防止出现同一个目录下面缓存数据过多的情况。

1.64 使用 ThinkPHP 的过程中为什么总是容易发生乱码？

A ThinkPHP 默认使用的是 UTF8 编码，确保你的编码设置正确并且和你的配置保持一致。

1.65 使用 ThinkPHP 开发一定要使用 UTF8 编码吗？

A ThinkPHP 使用 UTF8 编码只是参考目前的网站开发标准而采取的一种默认配置和建议，你完全可以通过配置更改你需要使用的编码格式，包括数据库编码、模板文件编码和输出编码，而且一旦配置系统还可以实现编码的自动转换。默认的配置下，以上三者的编码都是 UTF8 编码，对于相同的编码格式系统不会进行额外的编码转换过程。

1.66 如何用 S 方法删除缓存？


利用 S（‘name’,NULL); 可以删除标识为 name 的缓存。

1.67 如何设置永久缓存某个数据？


设置缓存有效期为-1 就可以永久缓存某个数据，例如 S（‘name’,$data,-1);

1.68 ThinkPHP 是怎么进行安全过滤的？


系统提供了多个层面的安全过滤机制，最大程度的保证了数据的安全。

首先，数据库底层驱动类已经对查询进行了安全过滤；

在模型里面可以定义自动验证来对提交的数据进行校验；

利用自动完成机制对非法篡改的数据进行重新写入；

利用字段自动过滤功能对写入数据库的字段进行过滤；

利用 Action 的 getParam 方法对提交的数据进行更严格的用户自定义过滤。

利用基于 RBAC 的权限验证机制防范没有授权的操作

ThinkPHP 提供了各种手段，但不代表系统会自动帮你完成各种安全过滤，完全要根据项目的要求进行合理的配置。

1.69 什么是 MVC？

 MVC 是一个设计模式，它强制性的使应用程序的输入、处理和输出分开。使用 MVC 应用程序被分成三个核心部件：模型（M）、视图（V）、控制器（C），它们各自处理自己的任务。

视图 ：视图是用户看到并与之交互的界面。对老式的 Web 应用程序来说，视图就是由 HTML 元素组成的界面，在新式的 Web 应用程序中，HTML 依旧在视图中扮演着重要的角色，但一些新的技术已层出不穷，它们包括 Adobe Flash 和象 XHTML，XML/XSL，WML 等一些标识语言和 Web services。如何处理应用程序的界面变得越来越有挑战性。MVC 一个大的好处是它能为你的应用程序处理很多不同的视图。在视图中其实没有真正的处理发生，不管这些数据是联机存储的还是一个雇员列表，作为视图来讲，它只是作为一种输出数据并允许用户操纵的方式。

模型 ：模型表示企业数据和业务规则。在 MVC 的三个部件中，模型拥有最多的处理任务。例如它可能用象 EJBs 和 ColdFusion Components 这样的构件对象来处理数据库。被模型返回的数据是中立的，就是说模型与数据格式无关，这样一个模型能为多个视图提供数据。由于应用于模型的代码只需写一次就可以被多个视图重用，所以减少了代码的重复性。

控制器 ：控制器接受用户的输入并调用模型和视图去完成用户的需求。所以当单击 Web 页面中的超链接和发送 HTML 表单时，控制器本身不输出任何东西和做任何处理。它只是接收请求并决定调用哪个模型构件去处理请求，然后确定用哪个视图来显示模型处理返回的数据。

现在我们总结 MVC 的处理过程，首先控制器接收用户的请求，并决定应该调用哪个模型来进行处理，然后模型用业务逻辑来处理用户的请求并返回数据，最后控制器用相应的视图格式化模型返回的数据，并通过表示层呈现给用户。

1.70 如何快速架构项目？

 首先将下载的 ThinkPHP 放在你的网站根目录下面，接着在网站根目录里面建立一个 index.php(不一定非要是 index.php 也可以是别的)，文件写入如下代码

```
define('THINK_PATH','./ThinkPHP/');
define('APP_NAME','MyApp');
define('APP_PATH','./MyApp');
require(THINK_PATH."/ThinkPHP.php");
App::run();
```

其中“.”代表 index.php 文件所在的路径，然后浏览器输入

http://localhost/index.php

运行后就会发现系统自动在网站根目录下面创建了 MyApp 文件夹，这只是一个例子，大家可以根据自己的需要对路径设置稍作修改就可以了。

1.71 为何无法更新数据表字段？

 用 ThinkPHP 的 save 方法更新数据，但是无法完成字段更新。这个时候可以删除字段缓存文件后重新测试，字段缓存文件位于 Runtime/Data/_fields/目录。

1.72 用 M 方法或者 D 方法实例化模型有什么区别？

A 简单说使用 M 方法的话，是不需要定义对应的模型类的（即使有定义也不会读取），通常这样的模型仅能调用一些系统基础模型类 Model 类里面的一些方法。而使用 D 方法实例化模型的话，必须有对应的模型类文件，可以调用一些模型自定义的方法或者属性，另外在 Mode 对应文件里面有自动验证或者函数之类的业务逻辑也必需用 D。

再打个比方说 M 是刚安装好的操作系统，只有系统自带的应用 还没有自己安装的应用（所以只能调用内置的 Model 提供的属性和方法）D 是已经安装了很多的第三方的应用程序 使用起来更丰富一些（可以调用模型类自己定义的属性和方法）但是很明显，安装了很多的第三方应用后系统性能降低了，执行变慢了，但是功能显然强大了。

1.73 修改了 Common 函数文件，怎么运行的时候没有任何变化？

A 修改了 Common 函数文件后，需要删除 Runtime 下面的缓存文件~app.php 才能生效，如果需要经常修改 Common 函数文件，请开启调试模式，系统就不会自动生成项目编译缓存文件。

1.74 如何添加自己的函数库？

A 放在项目目录下面的 Common/common.php 系统会自动加载该函数，但是修改 common.php 之后记得一定要删除下项目编译缓存文件~app.php。

1.75 如何更新同字段名的多条记录？

用 $_post[‘字段名’] 将得到一个数组，然后循环更新，参照代码

```
$M = M("Config");
for($i = 0;$i < count($_POST["id"]); $i++) {
$data["id"] = $_POST["id"][$i];
$data["body"] = $_POST["body"][$i];
$M->save($data);
}
```

1.76 为何 RBAC 改了路径就没有权限了？

RBAC::AccessDecision() 这个方法是权限判断的，默认是读取当前项目名称，项目名称又是你自己初始开设的项目目录，当提交 RBAC 时候，在数据表中有录入的初始的项目名称，现在又改项目目录名称了，查询对比不符，所以无权限。

参照代码：**RBAC::AccessDecision('现在的项目录名称');** 或者在数据表 Node 里找到你之前的项目目录名称，改成现在的项目名称。

1.77 为什么$this->error() 和$this->success()跳转同一个模板文件？

A Thinkphp 的默认配置错误和成功是一个模板，可以在配置里面添加

'TMPL_ACTION_ERROR' => 'Public:error' // 默认错误跳转对应的模板文件

'TMPL_ACTION_SUCCESS' => 'Public:success' //默认成功跳转对应的模板文件

1.78 怎么输出原生查询的结果？

A 如果使用了 ThinkPHP 的原生查询 Query 方法的话，返回的结果和 select 方法一样，是返回数据集而不是数据，因为是一个二维数组，所以在输出的时候应该使用 volist 模板标签输出。

1.79 如何获得上一步插入记录的 id？

A ThinkPHP 模型类的 add()方法返回值就是上一步插入数据的 id

或者调用模型类的 getLastInsID 方法

2 开发技巧

以下是我们整理的在使用 ThinkPHP 开发过程中的一些实用技巧，如果没有特殊的说明，均在 2.*版本中有效。

2.1 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

2.2 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~

2.3 定义实际的数据库名称

如果需要操作的某些数据表还存在跨库的情况，并且不是配置文件中定义的当前数据库，则可以在模型类中加上数据库的定义：

```
protected $dbName = '数据库名称';
```

注意，要确保当前数据库用户有跨库操作的权限。

2.4 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

2.5 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();
```

2.6 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

2.7 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。

2.8 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'");
```

2.9 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。

2.10 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。

2.11 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。

2.12 快速切换到其他的数据库

从 2.1 版本开始，不用继承高级模型类也可以在操作过程中切换不同的数据库了。2.1 版本的模型用很简洁的方式实现了数据库的切换，用法很简单， 只需要调用 Model 类的 db 方法，用法：

```
Model->db("数据库编号","数据库配置信息");
```

数据库编号用数字格式，对于已经调用过的数据库连接，是不需要再传入数据库连接信息的，系统会自动记录。

数据库配置信息采用 DSN 字符串方式设置，格式采用：

数据库类型://用户名:密码@数据库地址:数据库端口/数据库名称

Db 方法调用后返回当前的模型实例，不需要和高级模型中的切换数据库方法一样，需要先添加然后再切换过去，直接可以继续进行其他操作，所以该方法在查询的过程中动态切换，例如：

```
$this->db(1, "mysql://root:123456@localhost:3306/test")->query("查询 SQL");
```

该方法添加了一个编号为 1 的数据库连接，并自动切换到当前的数据库连接。

当第二次切换到相同的数据库的时候，就不需要传入数据库连接信息了，可以直接使用：

```
$this->db(1)->query("查询 SQL");
```

如果需要切换到默认的数据库连接，只需要调用：

```
$this->db(0);
```

2.13 利用别名快速加载类库

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。

定义之后，我们可以直接使用 myClass 和 myUtil 类，例如：

$class = new myClass();

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

2.14 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。

如何定义别名之前已经提过了，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，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.',

这样定义后，所有位于系统基类库 Think/Util/和项目应用类库 Lib/Common/ 下面的类库也会自动加载。

2.15 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
```

2.16 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

2.17 如何在表单里面隐藏字段名称

Thinkphp 的字段映射功能可以让你在表单中隐藏真正的数据表字段，而不用担心放弃 TP 的自动创建表单对象的功能，假设我们的 User 表里面有 username 和 email 字段，我们需要映射成另外的字段，通过定义模型的$_map 属性即可解决，定义方式如下：

```
class UserModel extends Model{
    protected $_map = array(
        'name' => 'username',
        'mail' => 'email',
    );
}
```

这样，在表单里面就可以直接使用 name 和 mail 名称作为表单数据提交了。在保存的时候会字段转换成定义的字段映射。

2.18 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
);
$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。

2.19 模型不需要数据库怎么定义

默认情况下只要定义了模型就会连接数据库，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。

2.20 如何实现延迟更新

要实现延迟更新，模型类必须继承高级模型类

```
class UserModel extends AdvModel{
}
```

延迟更新可以调用高级模型类的 setLazyInc 和 setLazyDec 方法，例如对用户的登录次数每 1 小时延迟一起更新：

```
$User = D('User');
// 对用户的登录次数进行延迟更新，每小时更新
$User->setLazyInc('login_count', 'id=5',1,3600);
```

如果没有继承高级模型类也可以采用动态模型处理

```
$User = D('User');
// 对用户的登录次数进行延迟更新，每小时更新一次
$User->switchModel('Adv')->setLazyInc('login_count', 'id=5',1,3600);
```

延迟更新的字段必须是整形字段，通常用于阅读数、评论数和登录数之类需要频繁更新的字段，而延迟更新可以在对数据的实时要求不高的情况下缓解数据库的写压力。

2.21 如何避免某个字段被修改？

ThinkPHP 的高级模型提供了只读字段的功能，可以防止某个字段被用户提交的数据更改。

```
class UserModel extends AdvModel{
protected $readonlyField = array('read_only_field1', 'read_only_field2');
}
```

所有在 readonlyField 属性中定义的字段将不会被更改。

2.22 如何使用乐观锁功能

ThinkPHP 也可以支持乐观锁机制，要启用乐观锁，只需要继承高级模型类并定义模型的 optimLock 属性，并且在数据表字段里面增加相应的字段就可以自动启用乐观锁机制了。默认的 optimLock 属性是 lock_version，也就是说如果要在 User 表里面启用乐观锁机制，只需要在 User 表里

面增加 lock_version 字段，如果有已经存在的其它字段作为乐观锁用途，可以修改模型类的 optimLock 属性即可，下面的示例修改乐观锁字段为 version：

```
class UserModel extends AdvModel{
protected $optimLock = 'version';
}
```

设置了乐观锁之后，我们就可以在模型的 add 和 save 方法中使用乐观锁功能了。每次写入或者更新数据，乐观锁对应的版本字段会自动比较和增加，如果发现记录在保存之前已经被更新过，会返回“记录已经更新”的错误提示，判断方法如下：

```
$User = D('User');
$User->create();
$result = $User->save();
if( false === $result){ // 错误捕获
    $this->error($User->getError());
}else{
    $this->success('用户记录更新成功');
}
```

如果存在 optimLock 属性对应的字段，但是需要临时关闭乐观锁机制，把 optimLock 属性设置为 false 就可以了。

```
class UserModel extends AdvModel{
```

```
protected $optimLock = false;
}
```

2.23 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 方式提交

isPost 是否是 POST 方式提交

isPut 是否是 PUT 方式提交

isDelete 是否是 DELETE 方式提交

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。

2.24 如何隐藏 URL 地址里面的 index.php?

去掉 URL 里面的 index.php 是为了 SEO 的需要，需要服务器开启 URL_REWRITE 模块。

以 Apache 为例，下面的配置过程可以参考下：

1、httpd.conf 配置文件中加载了 mod_rewrite.so 模块

2、AllowOverride None 将 None 改为 All

3、确保 URL_MODEL 设置为 2

4、把.htaccess 文件放到入口文件的同级目录下，其中添加下面内容：

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

2.25 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。

2.26 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：

2.0 版本使用如下定义（泛路由定义）

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

2.1 以上版本使用如下定义（正则路由定义）

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

然后就可以使用

http://serverName/User/3

访问了，等效于访问

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

2.27 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'

2.28 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。

2.29 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

2.30 如何在模板文件中使用运算符

TP 内置的模板引擎可以支持运算符的使用，只不过需要注意一些使用上的事项，用法很简单，例如：

{$a+$b+$c}

{$a*$b+$c}

{$a*$b+10}

上面的用法都是正确的，并且可以支持包括“+”、“-”、“*”、“/”、“%”在内的运算符，但当我们使用模板变量的时候，不能再使用点语法和常规的函数用法，例如：

{$user.score+10} 是错误的

{$user['score']+10} 是正确的

{$user['score']*$user['level']} 正确的

{$user['score']|myFun*10} 错误的

{$user['score']+myFun($user['level'])} 正确的

2.31 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 JS 代码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'=>'}>'

这样就和 JS 代码区别开来了。

2.32 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。

2.33 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
```

2.34 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/

2.35 在 Nginx 的下如何支持 PATHINFO

在 Nginx 低版本中，是不支持 PATHINFO 的，但是可以通过下面的方式解决：

首先在 nginx 中配置转发规则：

```
location / { // .....省略部分代码
  if (!-e $request_filename) {
  rewrite  ^(.*)$  /index.php?s=$1  last;
  break;
   }
```

}

然后，修改项目配置文件，设置 URL 模式为重写模式

'URL_MODEL' => 2

就可以通过：

http://serverName/User/add

来访问了，其实内部是转发到了 ThinkPHP 提供的兼容模式的 URL，利用这种方式，可以解决其他不支持 PATHINFO 的 WEB 服务器环境。

2.36 如何在 TP 中支持 Amf 开发

ThinkPHP 可以结合 ZendAMF 直接支持 Amf 开发，和 Flash 进行通讯。首先，我们需要设置当前运行模式为 Amf 模式，在入口文件中增加下面代码：

```
define('THINK_MODE', 'amf');  // 采用 Amf 运行模式运行
```

然后在项目配置文件中定义

```
'APP_AMF_ACTIONS'=>'Index,User,Shop...'  // 定义 AMF 模式的模块列表
```

只有在 APP_AMF_ACTIONS 中定义的模块才能在 Amf 模式中调用到。

最后一步就是在你的 Flash 客户端或者 AS 脚本中修改 Amf 的网关地址为当前项目的入口地址即可。

注意： Amf 扩展模式并非内置模式，你可能需要首先下载 Amf 模式扩展。

2.37 如何在 TP 中支持 Phprpc 开发

首先，我们设置当前运行模式为 Phprpc 模式：

```
define('THINK_MODE', 'phprpc');  // 采用 Phprpc 运行模式运行
```

然后在项目配置文件中定义

```
'APP_PHPRPC_ACTIONS'=>'Index,User,Shop...'  // 定义 PHPRPC 模式的模块列表
```

只有在 APP_PHPRPC_ACTIONS 中定义的模块才能在 PHPRPC 模式中调用到。

注意：Phprpc 扩展模式并非内置模式，你可能需要首先下载 Phprpc 模式扩展。

如何在 Action 方法中返回数据：

返回参数统一都是用 return；

返回字符串可以用 echo；

返回异常可以直接抛出异常 throw new Exception('string', 1);

2.38　利用分组的二级域名部署功能

2.1 版本开始，ThinkPHP 支持分组的二级域名部署功能。

只需要在配置文件中增加二级域名规则：

```
'APP_GROUP_LIST' => 'Home,Test,Admin', // 设置项目的分组列表
'APP_SUB_DOMAIN_DEPLOY' => 1, // 开启二级域名部署
// 定义子域名部署规则 定义格式：
// '子域名'=>array('分组名/[模块名]','var1=a&var2=b');
'APP_SUB_DOMAIN_RULES' => array(
    'admin' => 'Admin/', // admin 域名指向 Admin 分组
```

```
    'test' => 'Test/', // test 域名指向 Test 分组
)
```

然后只需要把子域名指向项目目录或者配置 Apache 即可。

2.39 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。

2.40 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

2.41 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

)

2.42 如何在页面输出__PUBLIC__

我们知道，在模板输出的时候，类似 __PUBLIC__、 __URL__和__APP__之类的字符串会被作为特殊字符串被自动替换，那么我们如何才能在页面实际输出这些字符串本身呢。

同样的办法，我们借助动态增加特殊字符串替换的方式来解决。在项目配置文件中配置

```
'TMPL_PARSE_STRING' => array(
    '@PUBLIC@' => '__PUBLIC__', // 增加__PUBLIC__ 输出定义
    '@URL@' => '__URL__', // 增加__URL__输出定义
    '@APP@' => '__APP__', //增加__APP__输出定义
)
```

定义的时候，前面可以随意设置，只要确保不容易冲突即可。经过配置之后，我们就可以直接在模板文件中采用 @PUBLIC@、 @URL@ 和@APP@这些字符串来输出__PUBLIC__、 __URL__和__APP__本身了。

2.43 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。

2.44 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');
```

2.45 如何定制网站的错误页面

默认情况下，ThinkPHP 在发生错误的时候，显示的是系统默认的错误页面，正式上线的时候，为了统一用户体验，我们可以定制自己的错误页面，通常有两种方法：

1、重新定义系统错误页面模板

系统默认的错误模板位于：ThinkPHP/Tpl/ThinkException.tpl.php

我们只需要在项目中修改 **TMPL_EXCEPTION_FILE** 配置参数重新指定错误模板即可。

```
'TMPL_EXCEPTION_FILE'=>'./App/Tpl/Public/error.tpl.php'  // 定义公共错误模板
```

注意错误模板的路径是基于入口文件的相对地址或者使用服务器的绝对地址，错误模板中可以使用的变量有：

$e['file'] 异常文件名

$e['line'] 异常发生的文件行数

$e['message'] 异常信息

$e['trace'] 异常的详细 Trace 信息

因为异常模板使用的是原生 PHP 代码，所以还可以支持任何的 PHP 方法和系统变量使用。

2、设置错误重定向页面

如果想网站发生错误的时候重定向到一个指定的 URL 而不是读取错误模板，我们还可以直接设置 ERROR_PAGE 参数。

```
'ERROR_PAGE'=>'/Public/error.html'  // 定义错误跳转页面 URL 地址
```

注意 ERROR_PAGE 所指向的页面不能再使用异常的模板变量了。

2.46 改变运行时间的显示位置

默认情况下，系统的运行时间显示位于 Html 页面的最后，如果希望调整显示位置，可以在模板文件中相应的位置加上下面的字符串：

{__RUNTIME__}

在输出的时候会在该位置自动替换成运行时间的显示信息。

2.47 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));
```

2.48 如何支持 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

新版本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。

2.49 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{}
```

.....

2.50 如何实现上传文件子目录保存

系统支持两种方式的文件上传子目录保存，包括哈希子目录和日期子目录方式。设置上传类的subType 属性即可，默认是 hash（哈希）方式，如果是日期子目录方式，设置为 date 即可。下面是参考代码：

```
import("ORG.Net.UploadFile");
$upload = new UploadFile(); // 实例化上传类
$upload->savePath = './Uploads/';  // 设置上传目录
$upload->autoSub = true;  // 开启文件子目录保存
$upload->subType = 'hash';  // 设置哈希子目录方式
如果是哈希方式，我们还可以设置子目录层次，默认是一层
$upload->hashLevel = 2; // 设置二级哈希子目录保存
如果是日期方式，我们还可以设置日期格式，默认是 Ymd
$upload->dateFormat = 'Y-m-d'; // 设置日期目录格式
```

2.51 图片上传如何实现自动缩略图

首先是实例化上传类

```
import("ORG.Net.UploadFile");
```

```
$upload = new UploadFile(); // 实例化上传类
$upload->savePath = './Uploads/';  // 设置上传目录
$upload->allowExts  = array('jpg', 'gif', 'png', 'jpeg'); // 设置图片上传类型
$upload->thumb = true;  // 开启缩略图功能
$upload->thumbPrefix = 'b_,s_';  // 设置缩略图前缀
$upload->thumbMaxWidth = '200,50';  // 设置缩略图最大宽度 多个用逗号分隔
$upload->thumbMaxHeight = '200,50';  // 设置缩略图最大高度 多个用逗号分隔
```

设置好上传的参数后，就可以调用 UploadFile 类的 upload 方法进行附件上传，如果失败，返回 false，并且用 **getErrorMsg** 方法获取错误提示信息；如果上传成功，可以通过调用 **getUploadFileInfo** 方法获取成功上传的附件信息列表。

```
if(!$upload->upload()) { // 上传错误 提示错误信息
    $this->error($upload->getErrorMsg());
}else{ // 上传成功 获取上传文件信息
    $info =  $upload->getUploadFileInfo();
 }
```

注意：其中生成缩略图功能需要 Image 类的支持。

2.52 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。

2.53 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。

2.54 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

2.55 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']" />

3 推荐阅读

3.1 .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。

3.1.1 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！"

3.1.2 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

3.1.3 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。

3.1.4 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
```

3.1.5 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
```

3.1.6 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

3.1.7 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]