ThinkPHP 1.5
错误、调试和日志

编写：ThinkPHP 文档组

最后更新：2008-12-16

目录

1 概述

本文系统的描述了 ThinkPHP 中的异常和错误处理，调试模式的相关使用，以及日志的处理。

2异常处理

和一般的异常处理不同，ThinkPHP 抛出的不是单纯的错误信息，而是一个人性化的错误页面。

ThinkPHP 定义了 throw_exception 方法，用于手动抛出异常。同样也可以使用 throw 关键字来抛出异常。

throw_exception 方法支持三个参数：

$msg 异常信息，必须

$type 异常类型，即异常类的名称，默认是系统异常基础类 ThinkException

$code 异常代码 默认为 0

如果指定的异常类型不存在，系统自动调用 halt 方法直接输出异常信息文字，而不输出异常详细信息。

下面是一些使用例子：

```
throw_exception('新增失败');
```

```
throw_exception('信息录入错误','InfoException');
```

异常页面的模板是可以修改的，通过设置 EXCEPTION_TMPL_FILE 配置参数来修改系统默认的异常模板文件，如果没有定义，则采用系统内置的异常模板文件，该模板文件位于 Think.Exception 目录下面的 ThinkException.tpl.php 文件。

另外一种方式是配置 ERROR_PAGE 参数，把所有异常和错误都指向一个统一页面，从而避免让用户看到异常信息，通常在部署模式下面使用。

如果需要，我们建议在项目的类库目录下面增加 Exception 目录用于专门存放异常类库，以更加精确地定位异常。

3 调试模式

在开启了框架的调试模式之后，我们会看到更加详细的错误信息，调试模式的作用在于显示或者记录了更多的日志信息，帮忙我们在项目开发过程中快速定位和解决问题。

开启调试模式很简单，只要在项目配置文件里面设置

```
<?php
return array(
```

```
'DEBUG_MODE' => true,
);
```

开启调试模式之后，系统在运行的时候首先会检查项目是否有定义调试配置文件，如果没有定义则调用框架默认的调试配置文件里面的参数，这些是系统为调试模式预设的默认配置。系统的默认调试配置文件位于 ThinkPHP\Common\debug.php。在这个默认的调试配置文件里面，系统开启了日志记录、关闭了页面防刷新机制、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。如果你觉得默认的调试配置不符合你的项目调试需要，你还可以在项目里面定义调试配置文件。

4 调试配置

我们可以给项目单独定义调试配置文件，用于项目在调试模式下面的配置信息。

项目的调试配置文件位于配置目录 Conf 目录下面，文件名定义为 **debug.php**，格式和项目配置文件的定义方法完全相同。

调试配置文件仅仅在启用调试模式的情况下有效，一旦项目关闭调试模式，就依然会使用项目配置文件里面的配置信息。

需要注意的是，调试模式下面并不是说项目只会加载调试配置文件，项目配置文件依然会首先加载的，只不过调试配置文件里面存在和项目配置文件有冲突的情况下 ,会覆盖项目配置文件里面的相同参数。也就是说，和项目配置文件相同的参数可以不必在调试模式下面进行定义。在下面的情况下，你通常会考虑使用项目的调试配置文件：

✧ 调试模式需要连接不同的测试数据库

✧ 需要增加额外的调试配置信息

5日志处理

ThinkPHP 内置了日志处理功能，包括系统异常和错误和调试信息，以及 SQL 记录，日志文件分别对应为 WEB_LOG_ERROR 、WEB_LOG_DEBUG 和 SQL_LOG_DEBUG 三种类型，对应的日志文件名称为：

systemErr.log 主要用于 WEB_LOG_ERROR 类型日志

用于记录系统异常，通常为抛出异常或者捕获严重错误后自动记录

systemOut.log 主要用于 WEB_LOG_DEBUG 日志类型

用于记录调试信息和页面的一些非严重错误记录，调试信息一般为 system_out 方法写入。

systemSql.log 主要是用于 SQL_LOG_DEBUG 日志类型

记录执行过程中的 SQL 语句和执行时间，便于进行分析和优化。

设置 WEB_LOG_RECORD 为 true 就可以启用日志记录功能，日志文件的命名规则是前面增加日期前缀，原则上是一天的同类型的日志记录在一个文件里面，您可以随时查看日志文件,例如：

07_09_21_systemOut.log // 2007 年 9 月 21 日的错误日志文件

07_12_03_systemSql.log // 2007 年 12 月 3 日的 SQL 日志文件

07_02_03_systemErr.log // 2007 年 2 月 3 日的异常日志文件

可以设置 LOG_FILE_SIZE 参数来限制日志文件的大小，超过大小的日志会形成备份文件。备份文件的格式是在当前文件名前面加上备份的时间戳，例如：

1189571417-07_09_12_systemSql.log 备份的 SQL 日志文件

在系统的调试模式中，系统的所有异常和错误都会记录到系统日志中，在正式部署应用后，您可以关闭调试模式，这样系统就不会自动完成日志记录，除非你自己触发日志写入。

系统对每个项目单独记录日志，所以查看的时候请注意定位到某个项目目录下。

如果您的应用组件需要记录特殊的日志，也可以调用（或者扩展）该方法来完成。

6 错误记录

配置 WEB_LOG_RECORD 参数可以开启错误日志记录

```
'WEB_LOG_RECORD' => true,
```

错误日志文件位于项目的 Logs 目录下面，是按照日期分开文件存放的

命名格式是：日期+systemErr.log

例如：

07_02_03_systemErr.log　// 2007 年 2 月 3 日的错误日志文件

08_05_13_systemErr.log　// 2008 年 5 月 13 日的错误日志文件

在开启 WEB_LOG_RECORD 的情况下，系统会自动在 systemErr.log 日志文件里面记录相关的错误信息，例如：

[08-05-07 17:21:59]

注意：[2] Invalid argument supplied for foreach() e8a01c49e3bd6881d1526bce80cbcad7.php 第 7 行.

[08-05-07 17:23:50]

错误信息：[myApp Index index]模板不存在

错误页面：/myapp/index.php

错误类型：ThinkException

错误跟踪：

[08-05-07 17:23:05] D:\www\myApp\Temp\~runtime.php (9) View->fetch(in, , text/html, , 1)

[08-05-07 17:23:05] D:\www\myApp\Temp\~runtime.php (9) View->display(in, , text/html,)

[08-05-07 17:23:05] D:\www\myApp\Lib\Action\IndexAction.class.php (9) Action->display(in)

[08-05-07 17:23:05] D:\www\myApp\Temp\~runtime.php (9) IndexAction->index()

[08-05-07 17:23:05] D:\www\myApp\Temp\~runtime.php (9) App->exec()

[08-05-07 17:23:05] D:\www\myApp\index.php (12) App->run()

可以设置 LOG_FILE_SIZE 参数来限制日志文件的大小，超过大小的日志会形成备份文件。备份文件的格式是在当前文件名前面加上备份的时间戳，例如：

1189571417-07_09_12_systemErr.log 备份的错误日志文件

7SQL 记录

配置 SQL_DEBUG_LOG 参数可以开启 SQL 日志记录

```
'SQL_DEBUG_LOG' => true,
```

SQL 日志文件位于项目的 Logs 目录下面，是按照日期分开文件存放的，

命名格式是：日期+systemSql.log

例如：

07_02_03_systemSql.log // 2007 年 2 月 3 日的 SQL 日志文件

08_05_13_systemSql.log // 2008 年 5 月 13 日的 SQL 日志文件

在开启 SQL 日志记录的情况下，系统会在 systemSql.log 日志文件里面自动记录执行过的 sql 语句和执行时间，下面是记录的信息：

[08-05-07 17:04:01]

RunTime:0.000270s SQL = SELECT * FROM think_blog

[08-05-07 20:38:33]

RunTime:0.000528s SQL = select `id` from `think_cache` where `cachekey`='aaaa' limit 0,1

我们也可以手动写入 SQL 日志信息

8手动记录日志

通常日志文件的写入是自动完成的，如果我们需要在开发的过程中手动记录日志信息，可以使用 Log 类的方法来操作。日志文件的写入有两种方法：

一、使用 **Log::Write**($message,$type,$file) 静态方法

$message 是要记录的日志信息

$type 就是日志类型

$file 日志文件位置和名称，该参数可以改变系统默认的日志文件命名。

Write 方法把日志信息直接写入相关的日志文件里面。

二、使用 Log::record 和 Log::save 方法

Log::record($message,$type);

$message 是要记录的日志信息

$type 就是日志类型

注意，record 方法只是把日志信息保存到内存，并没有真正写入日志文件。

Log::save()

保存 Log::record 方法记录的日志信息到相关的日志文件。

例如：

Log::record('测试调试错误信息', WEB_LOG_DEBUG);

Log::record('调试的 SQL：'.$SQL, SQL_LOG_DEBUG);

Log::save();

假如之前没有任何日志文件的话，上面的代码就会在项目的 Logs 目录下面生成两个日志文件，分别是 08_07_02_systemOut.log 和 08_07_02_systemSql.log(假设调试当日是 2008-7-2)

9 输出变量

调试模式并不能完全满足我们调试的需要，有时候我们需要手动的输出一些调试信息。

除了本身可以借助一些开发工具进行调试外，ThinkPHP 还提供了一些内置的调试函数和类库。

输出某个变量是开发过程中经常会用到的调试方法，除了使用 php 内置的 var_dump 和 print_r 之外，ThinkPHP 框架内置了一个对浏览器友好的 var_dump 方法，用于输出变量的信息到浏览器查看。

dump($var, $echo=true, $label=null) //输出变量信息

例如：

```
$Blog = D("Blog");
$blog = $Blog->find(3);
dump($blog);
// 在浏览器输出的结果是
array(12) {
  ["id"] => string(1) "3"
  ["name"] => string(0) ""
  ["userId"] => string(1) "0"
  ["categoryId"] => string(1) "0"
  ["title"] => string(4) "test"
  ["content"] => string(4) "test"
  ["cTime"] => string(1) "0"
  ["mTime"] => string(1) "0"
  ["status"] => string(1) "0"
  ["readCount"] => string(1) "0"
```

["commentCount"] => string(1) "0"

["tags"] => string(0) ""

}

10 输出错误信息

我们可以使用下面的方法输出错误信息：

halt($msg) //输出错误信息，并中止执行

system_out($msg) //输出调试信息到日志文件

11 运行时间和内存占用

使用下面的方法可以很方便的获取某个区间的运行时间和内存占用情况

debug_start($label='') //记录调试开始时间

debug_end($label='') //输出调试范围运行时间（相同 label 属于一个调试范围）

例如：

```
debug_start('run');
$blog = D("Blog");
$blog->findAll();
debug_end('run');
```

会输出下面的运行信息：

Process run: Times 0.007730s Memories 76 k

如果要输出内存占用情况，需要服务器支持 memory_get_usage 方法

更高级的调试方法是使用 Debug 类

Debug::mark($name); // 标记一个调试位置

Debug::useTime($start,$end); // 返回区间所用的时间

Debug::useMemory($start,$end); // 返回区间所用的内存

例如，下面的例子展示了如何使用 Debug 类来进行查询性能比较和输出。

```
$db = D('Form');
import('ORG.Util.HashMap');
Debug::mark('Start');
$map1 = new HashMap();
$map1->put('id',array('in',array(1,2,4)));
$db->findAll($map1);
Debug::mark('HashMap');
$map2= new stdClass();
$map2->id = array('in',array(1,2,4));
$db->findAll($map2);
Debug::mark('StdClass');
$map3=array();
$map3['id'] = array('in',array(1,2,3));
$db->findAll($map3);
Debug::mark('Array');
$map4 = 'id in (1,2,4)';
$db->findAll($map4);
Debug::mark('String');
$db->query('SELECT * FROM __TABLE__ WHERE id in (1,2,4)');
$timer->mark('Sql');
echo "<br/>Start->Hashmap:".Debug::useTime('Start', 'HashMap');
echo "<br/>HashMap->StdClass:".Debug::useTime('HashMap','StdClass');
echo "<br/>StdClass->Array:".Debug::useTime('StdClass','Array');
echo "<br/>Array->String:".Debug::useTime('Array','String');
```

```
echo "<br/>String->Sql:".Debug::useTime('String','Sql');
```

12 SQL 调试

有些时候你需要知道数据查询执行的 SQL 语句是否出错，但是又不想开启 SQL 日志记录，可以使用模型的 SQL 调试方法 **getLastSql**，例如：

$User = D(“User”);

// 执行查询

$User->where(‘status>1’)->order(‘create_time desc’)->limit(10)->findAll();

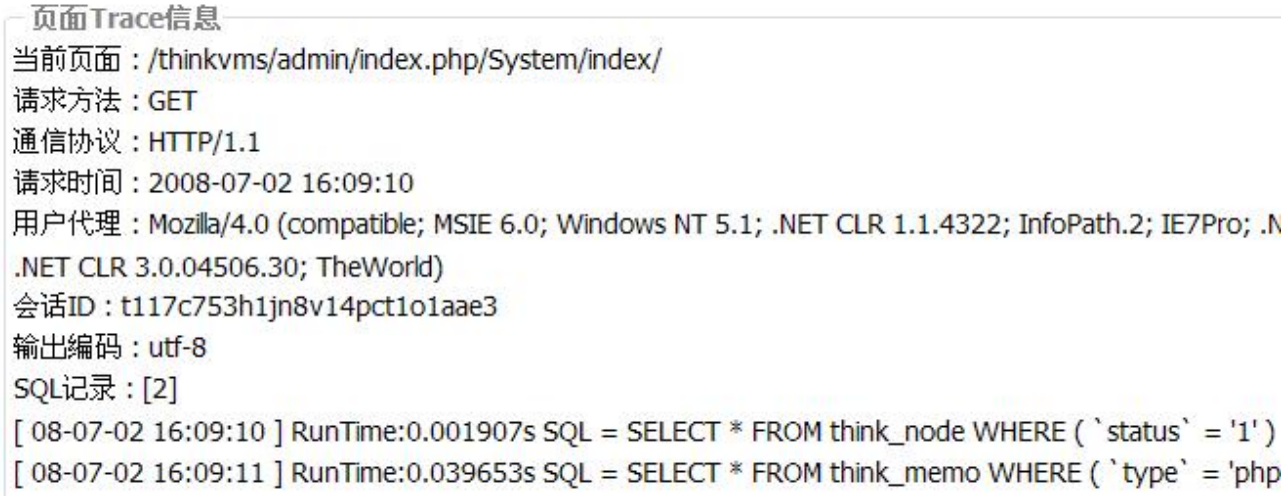
// 查看上次执行的 SQL 语句

echo $User->getLastSql();

通过查看和分析执行的 SQL 语句，可以准确的判断查询导致的错误原因。

13 页面 Trace

页面 Trace 功能是 ThinkPHP 自 1.0.3 版本后增加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、输出编码、模板编译、SQL 记录、错误记录。下图显示了页面 Trace 的显示信息



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));
```