本篇我们来全面了解下如何在ThinkPHP开发过程中进行调试工作。

**调试模式**

说到调试，我们可能第一个想到的就是调试模式，没错，ThinkPHP也有专门为开发过程而设置的调试模式，ThinkPHP的调试模式与众不同，会牺牲一定的执行效率，但带来的方便和除错功能非常值得。 我们强烈建议ThinkPHP开发人员在开发阶段始终开启调试模式（直到正式部署后关闭调试模式），方便及时发现隐患问题和分析、解决问题。[-more-]  
开启调试模式很简单，只需要在入口文件中增加一行常量定义代码：

1. <?php
2. //开启调试模式
3. define('APP\_DEBUG', true);
4. //加载框架入口文件
5. require './ThinkPHP/ThinkPHP.php';

复制代码

在完成开发阶段部署到生产环境后，只需要删除调试模式定义代码即可切换到部署模式。 调试模式的优势在于：

* 开启日志记录，任何错误信息和调试信息都会详细记录，便于调试；
* 关闭模板缓存，模板修改可以即时生效；
* 记录SQL日志，方便分析SQL；
* 关闭字段缓存，数据表字段修改不受缓存影响；
* 严格检查文件大小写（即使是Windows平台），帮助你提前发现Linux部署问题；
* 可以方便用于开发过程的不同阶段，包括开发、测试和演示等任何需要的情况，不同的应用模式可以配置独立的项目配置文件；

在开启调试模式的状态下，我们可以给项目设置不同的应用状态，并加载不同的项目配置文件，但是无论如何，都会首先导入框架默认的调试模式配置文件，该文件位于系统目录的Conf\debug.php。  
通常情况下，调试配置文件里面可以进行一些开发模式所需要的配置。例如，配置额外的数据库连接用于调试，开启日志写入便于查找错误信息、开启页面Trace输出更多的调试信息等等。  
如果没有配置应用状态，系统默认则默认为debug状态，也就是说默认的配置参数是：

1. 'APP\_STATUS' => 'debug', //应用调试模式状态

复制代码

如果检测到项目的配置目录中有存在debug.php文件，则会自动加载该配置文件，并且和系统项目配置文件以及系统调试配置文件合并，也就是说，debug.php配置文件只需要配置和项目配置文件以及系统调试配置文件不同的参数或者新增的参数。 如果想在调试模式下面增加应用状态，例如测试状态，则可以在项目配置文件中改变设置如下：

1. 'APP\_STATUS' => 'test', //应用调试模式状态

复制代码

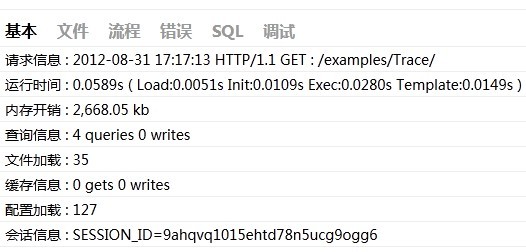
这样的话，系统会自动尝试加载项目配置目录下面的test.php 配置文件，可以在test配置文件中改变相关设置，例如改变测试数据库的连接信息等等。  
由于调试模式没有任何缓存，因此涉及到较多的文件IO操作和模板实时编译，所以在开启调试模式的情况下，性能会有一定的下降，但不会影响部署模式的性能。另外需要注意的是，一旦关闭调试模式，项目的调试配置文件即刻失效。

**页面Trace**

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工具和函数。例如，页面Trace功能就是ThinkPHP提供给开发人员的一个用于开发调试的辅助工具。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，并支持自定义显示。 页面Trace功能对调试模式和部署模式都有效，不过只能用于有页面输出的情况（如果你的操作没有任何输出，那么可能页面Trace功能对你帮助不大，你可能需要使用后面的调试方法）。但是在部署模式下面，显示的调试信息没有调试模式完整，通常我们建议页面Trace配合调试模式一起使用。  
要开启页面Trace功能，需要在项目配置文件中设置：

1. 'SHOW\_PAGE\_TRACE' =>true, // 显示页面Trace信息

复制代码

该参数默认为关闭，开启后并且你的页面有模板输出的话，页面右下角会显示ThinkPHP的LOGO：  
[http://www.thinkphp.cn/Uploads/editor/2012-09-24/505fec2939c30.jpg](http://www.thinkphp.cn/Uploads/editor/2012-09-24/505fec2939c30.jpg)  
我们看到的LOGO后面的数字就是当前页面的执行时间（单位是秒） 点击该图标后，会展开详细的页面Trace信息，如图：  
[](http://www.thinkphp.cn/Uploads/editor/2012-09-24/505fec4fbd353.jpg)  
页面Trace框架有6个选项卡，分别是基本、文件、流程、错误、SQL和调试，点击不同的选项卡会切换到不同的Trace信息窗口。  
基本：当前页面的基本摘要信息，例如执行时间、内存开销、文件加载数、查询次数等等。  
文件：详细列出当前页面执行过程中加载的文件及其大小。  
流程：会列出当前页面执行到的行为和相关流程（待完善）。  
错误：当前页面执行过程中的一些错误信息，包括警告错误。  
SQL：当前页面执行到的SQL语句信息。  
调试：开发人员在程序中进行的调试输出。  
页面Trace的选项卡是可以定制和扩展的，默认的配置为：

1. 'TRACE\_PAGE\_TABS'=>array(
2. 'base'=>'基本',
3. 'file'=>'文件',
4. 'think'=>'流程',
5. 'error'=>'错误',
6. 'sql'=>'SQL',
7. 'debug'=>'调试'
8. )

复制代码

也就是我们看到的默认情况下显示的选项卡，如果你希望增加新的选项卡：用户，则可以修改配置如下：

1. 'TRACE\_PAGE\_TABS'=>array(
2. 'base'=>'基本',
3. 'file'=>'文件',
4. 'think'=>'流程',
5. 'error'=>'错误',
6. 'sql'=>'SQL',
7. 'debug'=>'调试',
8. 'user'=>'用户'
9. )

复制代码

也可以把某几个选项卡合并，例如：

1. 'TRACE\_PAGE\_TABS'=>array(
2. 'base'=>'基本',
3. 'file'=>'文件',
4. 'think'=>'流程',
5. 'error|debug|sql'=>'调试',
6. 'user'=>'用户'
7. )

复制代码

我们把刚才的用户信息调试输出到用户选项卡，trace方法的用法如下：

1. trace($user,'用户信息','user');

复制代码

第三个参数表示选项卡的标识，和我们在TRACE\_PAGE\_TABS中配置的对应。  
默认情况下，页面Trace窗口显示的信息是不会保存的，如果希望保存这些trace信息，我们可以配置PAGE\_TRACE\_SAVE参数：

1. 'PAGE\_TRACE\_SAVE'=>true

复制代码

开启页面trace信息保存后，每次的页面Trace信息会以日志形式保存到项目的日志目录中，命名格式是： 当前日期\_trace.log，例如：

1. 12-06-21\_trace.log

复制代码

如果不希望保存所有的选项卡的信息，可以设置需要保存的选项卡，例如：

1. 'PAGE\_TRACE\_SAVE'=>array('base','file','sql');

复制代码

设置后只会保存base、file和sql三个选项卡的信息。

**Trace方法**

页面Trace只能用于有页面输出的情况，但是trace方法可以用在任何情况，而且trace方法可以用于AJAX等操作。  
Trace方法的格式： trace('变量','标签','级别','是否记录日志') 例如：

1. $info = '测试信息';
2. trace($info,'提示');

复制代码

如果希望把变量调试输出到页面Trace的某个选项卡里面，可以使用：

1. trace($info,'提示','user');

复制代码

表示输出到user选项卡，如果没有指定选项卡的话，默认会输出到debug选项卡。  
trace方法也可以直接抛出异常，如果是输出到ERR选项卡，并且开启

1. 'TRACE\_EXCEPTION'=>true

复制代码

的话，

1. trace($info,'错误','ERR');

复制代码

会抛出异常。  
有三种情况下，trace方法会记录日志：

* AJAX请求
* SHOW\_PAGE\_TRACE为false，也就是页面Trace关闭的情况下
* trace方法的第四个参数为true

在这种情况下，trace方法的第三个参数就表示记录的日志级别，通常包括：

1. 'ERR'  // 一般错误: 一般性错误
2. 'WARN'  // 警告性错误: 需要发出警告的错误
3. 'NOTIC'  // 通知: 程序可以运行但是还不够完美的错误
4. 'INFO'  // 信息: 程序输出信息
5. 'DEBUG'  // 调试: 调试信息
6. 'SQL'  // SQL：SQL语句

复制代码

**变量调试**

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 浏览器友好的变量输出  
用法 dump($var, $echo=true, $label=null, $strict=true)  
参数 var（必须）：要输出的变量，支持所有变量类型  
echo（可选）：是否直接输出，默认为true，如果为false则返回但不输出  
label（可选）：变量输出的label标识，默认为空  
strict（可选）：输出变量类型，默认为true，如果为false则采用print\_r输出  
返回值 如果echo参数为false 则返回要输出的字符串  
使用示例：

1. $Blog = D("Blog");
2. $blog = $Blog->find(3);
3. dump($blog);

复制代码

在浏览器输出的结果是：

1. array(12) {
2. ["id"]           => string(1) "3"
3. ["name"]         => string(0) ""
4. ["user\_id"]       => string(1) "0"
5. ["cate\_id"]   => string(1) "0"
6. ["title"]        => string(4) "test"
7. ["content"]      => string(4) "test"
8. ["create\_time"]        => string(1) "0"
9. ["update\_time"]        => string(1) "0"
10. ["status"]       => string(1) "0"
11. ["read\_count"]    => string(1) "0"
12. ["comment\_count"] => string(1) "0"
13. ["tags"]         => string(0) ""
14. }

复制代码

**性能调试**

开发过程中，有些时候为了测试性能，经常需要调试某段代码的运行时间或者内存占用开销，系统提供了G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。 例如：

1. G('begin');
2. // ...其他代码段
3. G('end');
4. // ...也许这里还有其他代码
5. // 进行统计区间
6. echo G('begin','end').'s';

复制代码

G('begin','end') 表示统计begin位置到end位置的执行时间（单位是秒），begin必须是一个已经标记过的位置，如果这个时候end位置还没被标记过，则会自动把当前位置标记为end标签，输出的结果类似于：

1. 0.0056s

复制代码

默认的统计精度是小数点后4位，如果觉得这个统计精度不够，还可以设置例如：

1. G('begin','end',6).'s';

复制代码

可能的输出会变成：

1. 0.005587s

复制代码

如果你的环境支持内存占用统计的话，还可以使用G方法进行区间内存开销统计（单位为kb），例如：

1. echo G('begin','end','m').'kb';

复制代码

第三个参数使用m表示进行内存开销统计，输出的结果可能是：

1. 625kb

复制代码

同样，如果end标签没有被标记的话，会自动把当前位置先标记位end标签。  
如果环境不支持内存统计，则该参数无效，仍然会进行区间运行时间统计。

**断点调试**

凭借强大的页面Trace信息功能支持，ThinkPHP可以支持断点调试功能。  
我们只需要在不同的位置对某个变量进行trace输出即可，例如：

1. $blog = D("Blog");
2. $vo = $blog->create();
3. trace($vo,'create vo');
4. $vo = $blog->find();
5. trace($vo,'find vo');

复制代码

**错误调试**

如果需要我们可以使用下面的方法输出错误信息并中断执行：

1. halt($msg)  //输出错误信息，并中止执行

复制代码

**模型调试**

在模型操作中 ，为了更好的查明错误，经常需要查看下最近使用的SQL语句，我们可以用getLastsql方法来输出上次执行的sql语句。例如：

1. $User = M("User"); // 实例化User对象
2. $User->find(1);
3. echo $User->getLastSql();

复制代码

输出结果是

1. SELECT \* FROM think\_user WHERE id = 1

复制代码

并且每个模型都使用独立的最后SQL记录，互不干扰，但是可以用空模型的getLastSql方法获取全局的最后SQL记录。

1. $User = M("User"); // 实例化User模型
2. $Info = M("Info"); // 实例化Info模型
3. $User->find(1);
4. $Info->find(2);
5. echo M()->getLastSql();
6. echo $User->getLastSql();
7. echo $Info->getLastSql();

复制代码

输出结果是

1. SELECT \* FROM think\_info  WHERE id = 2
2. SELECT \* FROM think\_user  WHERE id = 1
3. SELECT \* FROM think\_info  WHERE id = 2

复制代码

getLastSql方法只能获取最后执行的sql记录，如果需要了解更多的SQL日志，可以通过查看当前的页面Trace或者日志文件。  
  
注意：Mongo数据库驱动由于接口的特殊性，不存在执行SQL的概念，因此SQL日志记录功能是额外封装实现的，所以出于性能考虑，只有在开启调试模式的时候才支持使用getLastSql方法获取最后执行的SQL记录。vc