# Xiuno Framework 开发文档

作者: axiuno@gmail.com

时间：2011/10/3

# 概述

#### 1. Xiuno Framework 是什么?

Xiuno Framework 是一个NoSQL时代背景下产生的PHP框架，它很好的将DB(SQL + NoSQL)、Cache都封装起来，对外提供简单的几个接口，您不再需要分别记忆各种SQL语法和Cache操作的细节。存储层它支持MongoDB 、MySQL、PDO； Cache 层支持 Memcached、 TTServer、Redis、XCache、APC、EA等；

使用框架，应用的model 的CURD (Create、Update、Read、Delete)可以写成这样：

|  |
| --- |
| <?php  class blog extends base\_model {    function \_\_construct() {  parent::\_\_construct();  $this->table = 'blog';  $this->primarykey = array('blogid');  $this->cacheconf['cacheon'] = FALSE;  }  public function create($arr) {  $arr['blogid'] = $this->maxid('+1');  if($this->set($arr['blogid'], $arr)) {  $this->count('+1');  return $arr['blogid'];  } else {  $this->maxid('-1');  return FALSE;  }  }    public function update($blogid, $arr) {  return $this->set($arr['blogid'], $arr);  }    public function read($blogid) {  return $this->get($arr['blogid']);  }  public function \_delete($blogid) {  $return = $this->delete($blogid);  if($return) {  $this->count('-1');  }  return $return;  } |

如果你需要直接操作DB，您可以这样：

|  |
| --- |
| // 写一条记录  $db->set("user-userid-123", array('email'=>'a@gmail.com', 'password'=>'abcde'));  // 读一条记录  $user = $db->get('user-userid-123');  // 删除一条记录  $db->delete("user-userid-123"); |

当然，可以使用同样地规则操作 Cache:

|  |
| --- |
| // 写一条记录  $cache->set("user-userid-123", array('email'=>'a@gmail.com', 'password'=>'abcde'));  // 读一条记录  $user = $cache->get('user-userid-123');  // 删除一条记录  $cache->delete("user-userid-123"); |

这样，我们将接口统一起来，为您省去很多宝贵的学习的时间。是不是觉得这样操作数据太方便了！

本文档在以下篇幅将Xiuno Framework 缩写为 XF。

#### 2. Xiuno Framework 的来历

XF诞生于后Web2.0 时代，这个时代的应用向着海量数据，海量并发，复杂逻辑方向发展，SQL越来越难以满足新时期需求，面对苛刻的要求，开发者往往需要通过业务逻辑来拆分数据，存储到不同的服务器上，或者使用分库、分表、分区加上一层又一层的CACHE等手段，简直无不用其极，而需求则如天上浮云，变化多端，不得不反复迭代，令开发人员疲于奔命。

如果不想办法，这种现状只会越来越糟糕，那么多数据，那么多业务逻辑实实在在摆在那里，我们得想个办法解决它们，解放我们自己，于是有人提出了 NoSQL（Not Only SQL）, 在这个概念下， Xiuno Framework 诞生了。它将数据存储层(Storage: SQL+NoSQL)、缓存层(Cache: Memcached|TTServer|Redis|APC|...)封装起来，你只需要按照它的约定建立数据表结构和索引就可以轻松的像操作Cache一样操作数据库。通过它您也可以平滑的切换您的应用到NoSQL存储平台。

当然对数据层的封装只是它的重要特性之一，它还有其他优秀的特性，如果你愿意花上半个小时浏览完这片文档，我认为你将会收获很多，这毕竟是一个老程序员近十年的结晶。

# Hello World、

说了这么多，也许您已经急不可耐要看看它到底有多么方便了，好吧，那就从 Hello World 开始吧！

**假定：**

* + - 您本机已经架设好了php运行环境（如果您想快速体验，可以下载一个环境集成包： appserv）
    - 本机 web 目录: d:/www
    - 本机域名：http://localhost/

1. 下载http://www.xiuno.com/download/xiuno\_framework\_1.0.tar.gz

2. 解压到 d:/www/framework\_1.0

3. 建立一个文件 d:/www/test.php, 填入内容：

|  |
| --- |
| <?php  define('APP\_DEBUG', 1);  define('APP\_PATH', './');  define('FRAMEWORK\_PATH', APP\_PATH.'framework\_1.0');  include APP\_PATH.'conf.php';  include FRAMEWORK\_PATH.'core.php';  core::init();  echo 'Hello, World!';  ?> |

4. 浏览器访问： <http://localhost/test.php>， 输出结果：

Hello, World!

这段代码主要用来演示如何使用框架，本身没什么意义。真正要使用框架带来的好处还得继续往后看。

# Hello World 增强版: Hello, Blog!

刚才演示了一个非常简单的调用框架的实例，现在，我们开始使用框架做点有意义的Web 应用吧！

浏览器访问 <http://localhost/framework_1.0/gen/>

1. 系统会自动检测完您的环境，通过后，点击下一步，不通过的话请检查您的环境配置

2. 填写应用名称和路径，默认为 blog, d:/www/blog/, 点击下一步。

3. 填写MySQL数据库相关信息，默认是 localhost root root test。

3. 最后还需要您填写一个应用的数据表结构，默认是一个Blog 表，然后点击下一步。

4. 会询问您将该表那些字段设置为用户交互的控件，默认即可，下一步。

5. 自动生成完毕，访问： <http://localhost/blog/> ， 您还可以继续添加表，为应用增加更多的功能！

自动生成的代码在 d:/www/blog 下，这些代码的结构将在下面的章节有详细描述。这些代码只是一个比较粗糙的架子，您还需要为它添砖加瓦，继续打磨，所以您有必要熟悉下如何添砖加瓦和打磨。

# 命名规范

本框架遵守如下命名规范，同样我们期望您开发的应用也遵守如下规范，这样您会省去很多脑细胞花在记忆上。

1. 小写加下划线。
2. 文件夹名字，类名，避免使用复数。
3. 字段使用复数时候表示总数，比如： blogs, replies
4. 尽量连写，在大量同类变量时，使用前缀+下划线。比如：

|  |
| --- |
| userid, myspace, check\_subject(), check\_message(), check\_email(), |

1. 作用域越大的变量，名字含义越明确。越小的，可以采用更短的变量。比如作用域小的变量可以精简为如下：

|  |
| --- |
| for($i = 0; $i < $n; $i++) ...  foreach($arr as $k=>$v) ...  foreach($arrlist as $arr) ... |

总体说来是类似于UNIX C命名规范，区别于Java的骆驼命名规范和C++的匈牙利命名规范。

# 框架目录结构：

|  |
| --- |
| . [-] 根目录  [-] errorpage 异常错误时显示的页面  | |- exception.htm  | |- exception\_cmd.htm  | |- message.htm  [+] gen 代码自动生成工具，类似于MFC的应用程序向导  [-] lib 各种类库  | |- base\_control.class.php 重要！基础的 control 类，其他 control 继承它会获得一些方便。  | |- base\_model.class.php 重要！基础的 base 类，继承它会获得数据处理方面的很多方便  | |- **cache.interface.php** Cache的接口定义  | |- cache\_apc.class.php apc cache 的实现  | |- cache\_ea.class.php ea cache 的实现  | |- cache\_memcache.class.php memcache 的实现  | |- cache\_redis.class.php redis 的实现  | |- cache\_xcache.class.php xcache 的实现  | |- check.class.php 表单常用格式的校验类  | |- **core.class.php**  重要！核心的静态类，包含了常用的一些方法，和全局环境的初始化方法  | |- **db.interface.php**  DB 的接口的实现  | |- db\_file.class.php 文件存储的实现  | |- db\_mongodb.class.php mongodb的实现  | |- db\_mysql.class.php mysql 的实现  | |- db\_pdo.class.php pdo 的实现  | |- db\_ttserver.class.php ttserver 的实现  | |- encrypt.func.php 加密解密方法  | |- form.class.php 表单控件的生成类  | |- image.class.php 图片类  | |- log.class.php 日志类  | |- misc.class.php 跟 core.class.php 类似，一些不太常用的静态方法  | |- template.class.php 编译模板类  | |- utf8.class.php utf-8 字符操作类  | |- xn\_exception.class.php 异常显示类，用来显示详细的错误异常信息  [+] test 测试代码，用来测试框架代码是否正确  [+] tool 各种工具，大部分为命令行下执行  |- **core.php** 框架核心文件，应用在使用时候包含此文件即可。 |

# MVC(Model View Control)

#### 什么是MVC？

如果您需要开发一个大型的项目，那么MVC这个概念有必要花一点点时间了解下，这里简单罗嗦下什么是MVC。

MVC 是 Model(数据模型), View(显示), Control(流程控制) 的缩写，实际上在最原始的从上至下的编程方式上，已经包含了这三部分，只是在剥离后，显得更加清晰，容易维护。MVC 在代码“展开”后依然是顺序执行。

在WEB编程过程中，可以狭隘的理解MVC为如下：

View: HTML 模板，可以直接为HTM文件，也可以为Smarty 类的编译模板，当然我们强烈建议您不要使用它。

Model: 数据存取，一般这里会调用DB类取数据。

Control: 所谓的业务逻辑了，就是什么时候去Model里面取数据，格式化后调用 View 中的模板显示给用户。

用图形来表示这一模型：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  | | --- | | View |  |  | | --- | | 用点点滴滴户 |  |  | | --- | | Control |  |  | | --- | | Model | |

而实际项目的过程中，往往超出了三层，会根据业务的需要加层。

#### 我们的Model

在我们的框架中，Model 层被分为了四层：我们将Cache，Storage 都封装到了Model 。放大我们的Model:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| |  | | --- | | Model |  |  | | --- | | DB + Cache: 优先从Cache 取数据 |  |  | | --- | | Cache: Memcached TTServer Redis ... |  |  | | --- | | DB: mysql pdo mongodb ... |  |  | | --- | | DB + Cache + unique: 防止重复读取某一条数据 | |

#### 一个 model 的范例

如果你需要写一个user\_model, 以下是一个范例：

|  |
| --- |
| <?php  class blog extends base\_model {    function \_\_construct() {  parent::\_\_construct();  $this->table = 'blog';  $this->primarykey = array('blogid');    // 指定是否开启 cache  $this->cacheconf['cacheon'] = FALSE;  }    /\*  $arr = array(  'subject'=>'aaa',  'message'=>'bbb',  'dateline'=>'bbb',  );  \*/  public function create($arr) {  $arr['blogid'] = $this->maxid('+1');  if($this->set($arr['blogid'], $arr)) {  $this->count('+1');  return $arr['blogid'];  } else {  $this->maxid('-1');  return FALSE;  }  }    public function update($blogid, $arr) {  return $this->set($arr['blogid'], $arr);  }    public function read($blogid) {  return $this->get($arr['blogid']);  }  public function \_delete($blogid) {  $return = $this->delete($blogid);  if($return) {  $this->count('-1');  }  return $return;  }    // ------------------> 杂项  public function check\_subject(&$subject) {  if(empty($subject)) {  return '标题不能为空。';  }  if(utf8::strlen($subject) > 200) {  return '标题不能超过 200 字，当前长度：'.strlen($subject);  }  return '';  }    public function check\_message(&$message) {  if(empty($message)) {  return '内容不能为空。';  }  if(utf8::strlen($message) > 2000000) {  return '内容不能超过200万个字符。';  }  return '';  }    // 用来显示给用户  public function format(&$blog) {  $blog['subject'] = htmlspecialchars($blog['subject']);  $blog['message'] = $blog['message'];  $blog['dateline'] = misc::date($blog['dateline']);  }  }  ?> |

在 base\_model 中提供了 get() set() delete() maxid() count() 五种公共方法，由配置文件和选项来决定从 cache 还是 db 中存取数据。

#### model 与数据表的约定

在这里，要特别说明一点，每个model 对应一张数据表；model里需要指定primarykey ；因为我们为了统一db 和 cache 的接口，约定 db 的表必须有一个主键，并且可以用主键取到唯一的行。主键最多允许三列。

比如如下建表语句为合法：

|  |
| --- |
| CREATE TABLE `xn\_user` (  `**userid**` int(11) unsigned NOT NULL auto\_increment, # 用户id  `regip` int(11) NOT NULL default '0', # 注册ip  `regdate` int(11) unsigned NOT NULL default '0', # 注册日期  `username` char(16) NOT NULL default '', # 用户名  `password` char(32) NOT NULL default '', # 密码 md5()  KEY username(`username`),  **PRIMARY KEY (`userid`)**  ) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8\_general\_ci;  CREATE TABLE `xn\_fuser` (  `**fid**` int(11) unsigned NOT NULL default '0', # fid  `**userid**` int(11) unsigned NOT NULL default '0', # 用户id  `regip` int(11) NOT NULL default '0', # 注册ip  `regdate` int(11) unsigned NOT NULL default '0', # 注册日期  `username` char(16) NOT NULL default '', # 用户名  `password` char(32) NOT NULL default '', # 密码 md5()  KEY username(`username`),  **PRIMARY KEY (`fid`, `userid`)**  ) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8\_general\_ci; |

# AOP（面向切面编程）

# URL规则

#### 规则说明

本架构只支持一种URL规则，以下URL含义是等价的：

* /demo/index.php?module=blog&action=list&page=1&userid=123 [1]
* /demo/index.php?blog-list-page-1-userid-123.htm [2]
* **/demo/?blog-list-page-1-userid-123.htm [3] 书写时用这种**
* /demo/blog-list-page-1-userid-123.htm [4] 开启url-rewrite，自动转换

默认书写遵照第三种写法，在开启URL-Rewrite 后，架构会自动通过编译模板将URL转换为第四种，当然也可以显示的调用 core::url('blog-list-page-1') 来得到URL。以上URL 在 core:init() 以后，$\_GET 会被重新初始化为：

|  |
| --- |
| $\_GET = array (  **'0'=>blog,**  **'1'=>'list',**  **'page'=>1,**  **'userid'=>123**  );  **// 获取$\_GET 中的值：**  echo core::gpc(0); // 结果：blog  echo core::gpc(1): // 结果：list  echo core::gpc('page'); // 结果：1  echo core::gpc('userid'): // 结果：123 |

#### 服务器配置 URL-Rewrite

这样我们只需要一条 url-rewrite 规则即可实现所有URL静态化，对于搜索引擎是比较有利的。

1. Apache 可以在根目录下建立 .htaccess 文件

|  |
| --- |
| <IfModule mod\_rewrite.c>  RewriteEngine on  **RewriteRule ^\.htm$ index.php?$1 [L]**  </IfModule> |

1. Ngix 修改 /usr/local/nginx/conf/nginx.conf

|  |
| --- |
| server {  location / {  **rewrite "/([^/]+).htm$" /index.php?$1;**  index index.html index.htm index.php;  root /data/wwwroot/xxx.com;  }  } |

含义：将目录下所有的 xxx.htm 转发到 index.php?xxx.htm

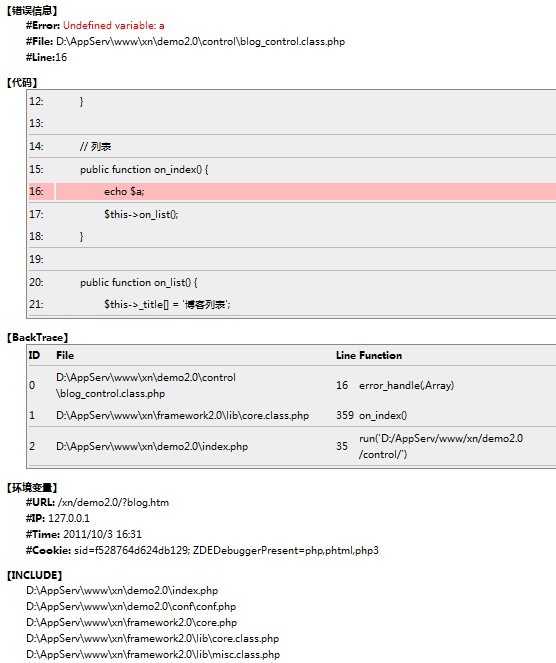
# 异常处理

框架自带了异常处理，用来跟踪详细的错误信息，方便开发者处理错误。

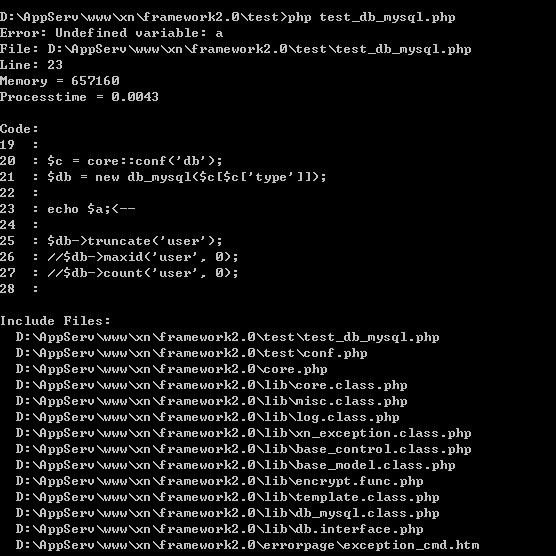
在define('APP\_DEBUG', 1); 以后，任何错误，包括一个Notice，都会触发异常。在上线以后一般设置为 0。

比如某个变量未定义： echo $a;

#### Web 方式执行时产生异常错误：



#### 命令行下执行时产生异常错误：



# 类库参考手册

#### template.class.php编译模板

##### 概述：

轻量级编译模板，用来代替smarty。

##### 支持的语法：

* 变量，数组

|  |
| --- |
| $username $arr[username] |

* 条件：

|  |
| --- |
| <!--{if}--> <!--{else}--> <!--{elseif}--> <!--{/if}--> |

* 循环：

|  |
| --- |
| <!--{loop $arrlist $arr}--> <!--{/loop}--> |

* 包含子模板：

|  |
| --- |
| <!--#include file="header.htm"--> |

* 路径转换（用来支持DreamWeaver 可视化编辑）：

|  |
| --- |
| <img convert="true" src="image/logo.gif" /> |

* BUTTON 美化（用来支持圆角Button）：

|  |
| --- |
| <input convert="true" type="button" value=" 创建博客 " class="button bigblue" onclick="" /> |

##### 示例1：

|  |
| --- |
| $t = new template();  $t ->assign('pages', $pages);  $t->assign('bloglist', $bloglist);  $t->display('blog\_list'); |

##### 示例2：

|  |
| --- |
| $t = new template();  $t->set\_path('view/'); // 如果需要转化路径，需要设置此项  $t ->displaytype = 'html'; // 输出格式是否为 json  $t ->urlrewrite = FALSE; // 是否开启 url-rewrite  $t ->assign('pages', $pages);  $t->assign('bloglist', $bloglist);  $t->assign\_value('blogs', 123); // 直接赋值  $t->display('blog\_list'); |

#### form.class.php 表单控件生成类

**用法：**

|  |
| --- |
| form:: get\_radio\_yes\_no('isadmin', 1);  form:: get\_checkbox\_yes\_no('isadmin', 1);  form::get\_radio('ishidden', array('否', '是'), 0);  form:: get\_select ('ishidden', array('否', '是'), 0);  form:: get\_checkbox('ishidden', array('否', '是'), 0);  form:: get\_text('email', 'xx@gmail.com', 150); |

#### utf8.class.php UTF-8 字符类

**用法：**

|  |
| --- |
| utf8::substr($str, $offset, $length = NULL);  utf8::is\_ascii($str);  utf8::strlen($str); |

#### log.class.php 日志类

用法：

|  |
| --- |
| log:: write($s, $logfile = ''); |

#### core.class.php

|  |
| --- |
| 获取$\_GET $\_POST $\_COOKIE $\_SERVER $\_REQUEST 中的值  core:: gpc($k, $var = 'G');  示例：  echo core::gpc('blogid'); // 默认从 $\_GET 中取数据  echo core::gpc('formhash', 'P'); // 从$\_POST 中取数据  echo core::gpc('sid', 'C'); // 从 $\_COOKIE 中取数据  获取配置文件中的 $conf 变量中的 $key 值  core:: conf($key, $default = NULL);  echo core::conf('sitename');  取得合法的URL，根据配置文件中 $conf['url-rewrite'] 设定返回URL格式。  core:: url($url);  echo core::url('abc'); |

#### misc.class.php

# 扩展类库

# 代码自动生成