[CodeIgniter源码分析之Loader.php 2](#_Toc454986470)

[CodeIgniter源码分析之Output.php 3](#_Toc454986471)

[CodeIgniter源码分析之Hook.php 3](#_Toc454986472)

[CodeIgniter源码分析之Config.php 3](#_Toc454986473)

[CodeIgniter源码分析之Benchmark.php 3](#_Toc454986474)

[CodeIgniter源码分析之Model.php 3](#_Toc454986475)

[CodeIgniter源码分析之Controller.php 3](#_Toc454986476)

[CodeIgniter源码分析之Exception.php 3](#_Toc454986477)

[CodeIgniter源码分析之Router.php 3](#_Toc454986478)

[CodeIgniter源码分析之URI.php 3](#_Toc454986479)

[CodeIgniter源码分析之Common.php 3](#_Toc454986480)

[CodeIgniter源码分析之CodeIgniter.php 3](#_Toc454986481)

[CodeIgniter源码分析之index.php 3](#_Toc454986482)

[CodeIgniter源码分析之Loader.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Loader.php%20–%20Calix的博客_files\CodeIgniter源码分析之Loader.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. /\*\*
3. \* Loader Class
4. \*
5. \* Loader组件在CI里面也是一个很重要的组件，功能也比较明了。
6. \* 如果已经阅读过Controller组件，会发现Controller组件的代码也只有十来行，但它却可以做很多事，一定程度上
7. \* 要归功于Loader组件这个好助手或者好基友。
8. \* 不过Loader组件的代码真的不少，主要以常用的几个方法以主线来探讨：model(),view(),library(),helper();
9. \*/
10. class CI\_Loader {
12. // All these are set automatically. Don't mess with them.
13. /\*\*
14. \* Nesting level of the output buffering mechanism
15. \*/
16. protected $\_ci\_ob\_level;
17. /\*\*
18. \* List of paths to load views from
19. \*/
20. protected $\_ci\_view\_paths = array();
21. /\*\*
22. \* List of paths to load libraries from
23. \*/
24. protected $\_ci\_library\_paths = array();
25. /\*\*
26. \* List of paths to load models from
27. \*/
28. protected $\_ci\_model\_paths = array();
29. /\*\*
30. \* List of paths to load helpers from
31. \*/
32. protected $\_ci\_helper\_paths = array();
33. /\*\*
34. \* List of loaded base classes
35. \*/
36. protected $\_base\_classes = array(); // Set by the controller class
37. /\*\*
38. \* List of cached variables
39. \*/
40. protected $\_ci\_cached\_vars = array();
41. /\*\*
42. \* List of loaded classes
43. \*/
44. protected $\_ci\_classes = array();
45. /\*\*
46. \* List of loaded files
47. \*/
48. protected $\_ci\_loaded\_files = array();
49. /\*\*
50. \* List of loaded models
51. \*/
52. protected $\_ci\_models = array();
53. /\*\*
54. \* List of loaded helpers
55. \*/
56. protected $\_ci\_helpers = array();
57. /\*\*
58. \* List of class name mappings
59. \*/
60. protected $\_ci\_varmap = array('unit\_test' => 'unit',
61. 'user\_agent' => 'agent');
63. /\*\*
64. \* Constructor
65. \*/
66. public function \_\_construct()
67. {
68. $this->\_ci\_ob\_level = ob\_get\_level();
69. $this->\_ci\_library\_paths = array(APPPATH, BASEPATH);
70. $this->\_ci\_helper\_paths = array(APPPATH, BASEPATH);
71. $this->\_ci\_model\_paths = array(APPPATH);
72. $this->\_ci\_view\_paths = array(APPPATH.'views/' => TRUE);
74. log\_message('debug', "Loader Class Initialized");
75. }
77. // --------------------------------------------------------------------
79. /\*\*
80. \* Initialize the Loader
81. \*/
82. public function initialize()
83. {
84. $this->\_ci\_classes = array();
85. $this->\_ci\_loaded\_files = array();
86. $this->\_ci\_models = array();
88. //这个is\_loaded方法就是在core/Common.php中定义的全局函数，在Loader组件还没有加载之前，由它来负责记录
89. //哪些核心类已经加载过，现在Loader组件要加载了，就把信息交给Loader组件，保存在Loader::$\_base\_classes中。
90. $this->\_base\_classes =& is\_loaded();
92. //自动加载，加载项是你在config/autoload.php中设置的。
93. $this->\_ci\_autoloader();
95. return $this;
96. }
98. // --------------------------------------------------------------------
100. /\*\*
101. \* Is Loaded
102. \*/
103. public function is\_loaded($class)
104. {
105. if (isset($this->\_ci\_classes[$class]))
106. {
107. return $this->\_ci\_classes[$class];
108. }
110. return FALSE;
111. }
113. // --------------------------------------------------------------------
115. /\*\*
116. \* Class Loader
117. \* $library为相应的类名，$params为实例化此类的时候可能要用到的参数，$object\_name为给这个类的实例自义定一个名字。
118. \*/
119. public function library($library = '', $params = NULL, $object\_name = NULL)
120. {
121. //如果是通过数组加载多个，把它拆开再调用本方法，其实它可以递归调用多维数组，不过没有这个必要。
122. if (is\_array($library))
123. {
124. foreach ($library as $class)
125. {
126. $this->library($class, $params);
127. }
129. return;
130. }
132. //接下来两个if都是关于合法性的判断。
133. if ($library == '' OR isset($this->\_base\_classes[$library]))
134. {
135. return FALSE;
136. }
138. if ( ! is\_null($params) && ! is\_array($params))
139. {
140. $params = NULL;
141. }
143. //真正把类加载进来的是下面这个方法。
144. $this->\_ci\_load\_class($library, $params, $object\_name);
145. }
147. // --------------------------------------------------------------------
149. /\*\*
150. \* Model Loader
151. \*/
152. public function model($model, $name = '', $db\_conn = FALSE)
153. {
154. //可以以数组形式同时加载多个$model
155. if (is\_array($model))
156. {
157. foreach ($model as $babe)
158. {
159. $this->model($babe);
160. }
161. return;
162. }
164. if ($model == '')
165. {
166. return;
167. }
169. $path = '';
171. //判断是否包含目录信息
172. if (($last\_slash = strrpos($model, '/')) !== FALSE)
173. {
174. $path = substr($model, 0, $last\_slash + 1);
176. $model = substr($model, $last\_slash + 1);
177. }
179. //如果没有给当前model定义名字，则以$model本身作为名字。
180. if ($name == '')
181. {
182. $name = $model;
183. }
185. //如果已经加载过此model，直接退出本函数。
186. if (in\_array($name, $this->\_ci\_models, TRUE))
187. {
188. return;
189. }
191. $CI =& get\_instance();
192. //如果加载的model名字与之前加载过的类有冲突，则报错。
193. if (isset($CI->$name))
194. {
195. show\_error('The model name you are loading is the name of a resource that is already being used: '.$name);
196. }
198. //model文件必段是全小写。
199. $model = strtolower($model);
201. foreach ($this->\_ci\_model\_paths as $mod\_path)
202. {
203. if ( ! file\_exists($mod\_path.'models/'.$path.$model.'.php'))
204. {
205. continue;
206. }
208. //如果要求同时连接数据库。则调用Loader::database()方法加载数据库类。
209. if ($db\_conn !== FALSE AND ! class\_exists('CI\_DB'))
210. {
211. if ($db\_conn === TRUE)
212. {
213. $db\_conn = '';
214. }
216. $CI->load->database($db\_conn, FALSE, TRUE);
217. }
219. //加载父类model。
220. if ( ! class\_exists('CI\_Model'))
221. {
222. load\_class('Model', 'core');
223. }
225. //引入当前model
226. require\_once($mod\_path.'models/'.$path.$model.'.php');
228. //把文件名的第一个字母大写作为类名，规定的命名规范。
229. $model = ucfirst($model);
231. $CI->$name = new $model();
233. //保存在Loader::\_ci\_models中，以后可以用它来判断某个model是否已经加载过。
234. $this->\_ci\_models[] = $name;
235. return;
236. }
238. show\_error('Unable to locate the model you have specified: '.$model);
239. }
241. // --------------------------------------------------------------------
243. /\*\*
244. \* Database Loader
245. \*/
246. public function database($params = '', $return = FALSE, $active\_record = NULL)
247. {
248. $CI =& get\_instance();
249. if (class\_exists('CI\_DB') AND $return == FALSE AND $active\_record == NULL AND isset($CI->db) AND is\_object($CI->db))
250. {
251. return FALSE;
252. }
254. require\_once(BASEPATH.'database/DB.php');
256. if ($return === TRUE)
257. {
258. return DB($params, $active\_record);
259. }
260. $CI->db = '';
262. $CI->db =& DB($params, $active\_record);
263. }
265. // --------------------------------------------------------------------
267. /\*\*
268. \* Load the Utilities Class
269. \*/
270. public function dbutil()
271. {
272. if ( ! class\_exists('CI\_DB'))
273. {
274. $this->database();
275. }
277. $CI =& get\_instance();
279. // for backwards compatibility, load dbforge so we can extend dbutils off it
280. // this use is deprecated and strongly discouraged
281. $CI->load->dbforge();
283. require\_once(BASEPATH.'database/DB\_utility.php');
284. require\_once(BASEPATH.'database/drivers/'.$CI->db->dbdriver.'/'.$CI->db->dbdriver.'\_utility.php');
285. $class = 'CI\_DB\_'.$CI->db->dbdriver.'\_utility';
287. $CI->dbutil = new $class();
288. }
290. // --------------------------------------------------------------------
292. /\*\*
293. \* Load the Database Forge Class
294. \*/
295. public function dbforge()
296. {
297. if ( ! class\_exists('CI\_DB'))
298. {
299. $this->database();
300. }
302. $CI =& get\_instance();
304. require\_once(BASEPATH.'database/DB\_forge.php');
305. require\_once(BASEPATH.'database/drivers/'.$CI->db->dbdriver.'/'.$CI->db->dbdriver.'\_forge.php');
306. $class = 'CI\_DB\_'.$CI->db->dbdriver.'\_forge';
308. $CI->dbforge = new $class();
309. }
311. // --------------------------------------------------------------------
313. /\*\*
314. \* Load View
315. \*
316. \* Loader::view();方法可以和Loader::file()方法一并来阅读，实质上它们都是调用了Loader::\_ci\_load();方法。
317. \*/
318. public function view($view, $vars = array(), $return = FALSE)
319. {
320. return $this->\_ci\_load(array('\_ci\_view' => $view, '\_ci\_vars' => $this->\_ci\_object\_to\_array($vars), '\_ci\_return' => $return));
321. }
323. // --------------------------------------------------------------------
325. /\*\*
326. \* Load File
327. \*/
328. public function file($path, $return = FALSE)
329. {
330. return $this->\_ci\_load(array('\_ci\_path' => $path, '\_ci\_return' => $return));
331. }
333. // --------------------------------------------------------------------
335. /\*\*
336. \* Set Variables
337. \*/
338. public function vars($vars = array(), $val = '')
339. {
340. if ($val != '' AND is\_string($vars))
341. {
342. $vars = array($vars => $val);
343. }
345. $vars = $this->\_ci\_object\_to\_array($vars);
347. if (is\_array($vars) AND count($vars) > 0)
348. {
349. foreach ($vars as $key => $val)
350. {
351. $this->\_ci\_cached\_vars[$key] = $val;
352. }
353. }
354. }
356. // --------------------------------------------------------------------
358. /\*\*
359. \* Get Variable
360. \*/
361. public function get\_var($key)
362. {
363. return isset($this->\_ci\_cached\_vars[$key]) ? $this->\_ci\_cached\_vars[$key] : NULL;
364. }
366. // --------------------------------------------------------------------
368. /\*\*
369. \* Load Helper
370. \*/
371. public function helper($helpers = array())
372. {
373. //Loader::\_ci\_prep\_filename()方法只是处理文件名，以返回正确的数组而已。
374. //默认helper的文件名是以\_helper为后缀，所以参数可以不用写\_helper后缀，当然写也不会出错，因为
375. //Loader::\_ci\_prep\_filename()会帮你处理掉。
376. foreach ($this->\_ci\_prep\_filename($helpers, '\_helper') as $helper)
377. {
378. //如果已经加载过此helper，则跳过。
379. if (isset($this->\_ci\_helpers[$helper]))
380. {
381. continue;
382. }
384. //helper的扩展。并非只有类可以扩展，函数也提供了扩展。注意这里是在APPPATH下。
385. $ext\_helper = APPPATH.'helpers/'.config\_item('subclass\_prefix').$helper.'.php';
386. //如果存在对些helper的扩展。
387. if (file\_exists($ext\_helper))
388. {
389. //先引入CI自带的helper，再引入我们写的扩展。
390. $base\_helper = BASEPATH.'helpers/'.$helper.'.php';
392. if ( ! file\_exists($base\_helper))
393. {
394. show\_error('Unable to load the requested file: helpers/'.$helper.'.php');
395. }
397. include\_once($ext\_helper);
398. include\_once($base\_helper);
400. $this->\_ci\_helpers[$helper] = TRUE;
401. log\_message('debug', 'Helper loaded: '.$helper);
402. continue;//继续下一个helper的引入。
403. }
405. //如果没有扩展的话，则分别从APPPATH和BASEPATH，即应用目录和系统目录下找到相应的helper。
406. //Loader::\_ci\_helper\_paths默认是APPPATH和BASEPATH两个目录。如果你要再添加新的目录路径，可以
407. //通过Loader::add\_package\_path()方法设置（model,library等同理）
408. foreach ($this->\_ci\_helper\_paths as $path)
409. {
410. if (file\_exists($path.'helpers/'.$helper.'.php'))
411. {
412. include\_once($path.'helpers/'.$helper.'.php');
414. $this->\_ci\_helpers[$helper] = TRUE;
415. log\_message('debug', 'Helper loaded: '.$helper);
416. break;
417. }
418. }
420. if ( ! isset($this->\_ci\_helpers[$helper]))
421. {
422. show\_error('Unable to load the requested file: helpers/'.$helper.'.php');
423. }
424. }
425. }
427. // --------------------------------------------------------------------
429. /\*\*
430. \* Load Helpers
431. \*/
432. public function helpers($helpers = array())
433. {
434. $this->helper($helpers);
435. }
437. // --------------------------------------------------------------------
439. /\*\*
440. \* Loads a language file
441. \*/
442. public function language($file = array(), $lang = '')
443. {
444. $CI =& get\_instance();
446. if ( ! is\_array($file))
447. {
448. $file = array($file);
449. }
451. foreach ($file as $langfile)
452. {
453. $CI->lang->load($langfile, $lang);
454. }
455. }
457. // --------------------------------------------------------------------
459. /\*\*
460. \* Loads a config file
461. \*/
462. //这里的config方法，实质是完完全全调用Config组件的load方法而已。
463. public function config($file = '', $use\_sections = FALSE, $fail\_gracefully = FALSE)
464. {
465. $CI =& get\_instance();
466. $CI->config->load($file, $use\_sections, $fail\_gracefully);
467. }
469. // --------------------------------------------------------------------
471. /\*\*
472. \* Driver
473. \*/
474. public function driver($library = '', $params = NULL, $object\_name = NULL)
475. {
476. if ( ! class\_exists('CI\_Driver\_Library'))
477. {
478. require BASEPATH.'libraries/Driver.php';
479. }
481. if ($library == '')
482. {
483. return FALSE;
484. }
486. if ( ! strpos($library, '/'))
487. {
488. $library = ucfirst($library).'/'.$library;
489. }
491. return $this->library($library, $params, $object\_name);
492. }
494. // --------------------------------------------------------------------
496. /\*\*
497. \* Add Package Path
498. \*/
499. public function add\_package\_path($path, $view\_cascade=TRUE)
500. {
501. $path = rtrim($path, '/').'/';
503. array\_unshift($this->\_ci\_library\_paths, $path);
504. array\_unshift($this->\_ci\_model\_paths, $path);
505. array\_unshift($this->\_ci\_helper\_paths, $path);
507. $this->\_ci\_view\_paths = array($path.'views/' => $view\_cascade) + $this->\_ci\_view\_paths;
509. $config =& $this->\_ci\_get\_component('config');
510. array\_unshift($config->\_config\_paths, $path);
511. }
513. // --------------------------------------------------------------------
515. /\*\*
516. \* Get Package Paths
517. \*
518. \*/
519. public function get\_package\_paths($include\_base = FALSE)
520. {
521. return $include\_base === TRUE ? $this->\_ci\_library\_paths : $this->\_ci\_model\_paths;
522. }
524. // --------------------------------------------------------------------
526. /\*\*
527. \* Remove Package Path
528. \*/
529. public function remove\_package\_path($path = '', $remove\_config\_path = TRUE)
530. {
531. $config =& $this->\_ci\_get\_component('config');
533. if ($path == '')
534. {
535. $void = array\_shift($this->\_ci\_library\_paths);
536. $void = array\_shift($this->\_ci\_model\_paths);
537. $void = array\_shift($this->\_ci\_helper\_paths);
538. $void = array\_shift($this->\_ci\_view\_paths);
539. $void = array\_shift($config->\_config\_paths);
540. }
541. else
542. {
543. $path = rtrim($path, '/').'/';
544. foreach (array('\_ci\_library\_paths', '\_ci\_model\_paths', '\_ci\_helper\_paths') as $var)
545. {
546. if (($key = array\_search($path, $this->{$var})) !== FALSE)
547. {
548. unset($this->{$var}[$key]);
549. }
550. }
552. if (isset($this->\_ci\_view\_paths[$path.'views/']))
553. {
554. unset($this->\_ci\_view\_paths[$path.'views/']);
555. }
557. if (($key = array\_search($path, $config->\_config\_paths)) !== FALSE)
558. {
559. unset($config->\_config\_paths[$key]);
560. }
561. }
563. $this->\_ci\_library\_paths = array\_unique(array\_merge($this->\_ci\_library\_paths, array(APPPATH, BASEPATH)));
564. $this->\_ci\_helper\_paths = array\_unique(array\_merge($this->\_ci\_helper\_paths, array(APPPATH, BASEPATH)));
565. $this->\_ci\_model\_paths = array\_unique(array\_merge($this->\_ci\_model\_paths, array(APPPATH)));
566. $this->\_ci\_view\_paths = array\_merge($this->\_ci\_view\_paths, array(APPPATH.'views/' => TRUE));
567. $config->\_config\_paths = array\_unique(array\_merge($config->\_config\_paths, array(APPPATH)));
568. }
570. // --------------------------------------------------------------------
572. /\*\*
573. \* Loader
574. \*/
575. protected function \_ci\_load($\_ci\_data)
576. {
577. //这里相当于把数组里面的元素拆开成变量。
578. foreach (array('\_ci\_view', '\_ci\_vars', '\_ci\_path', '\_ci\_return') as $\_ci\_val)
579. {
580. $$\_ci\_val = ( ! isset($\_ci\_data[$\_ci\_val])) ? FALSE : $\_ci\_data[$\_ci\_val];
581. }
583. $file\_exists = FALSE;
585. //当Loader::\_ci\_load()方法是通过Loader::file()调用的时候，则会有$\_ci\_path的值，如果是
586. //如果Loader::view()调用的话，则有$\_ci\_view的值。
587. //如果$\_ci\_path不为空，则说明当前要加载普通文件。
588. if ($\_ci\_path != '')
589. {
590. //普通文件。这里只是获得文件名，以便找不到报错时候只报文件名而已。
591. $\_ci\_x = explode('/', $\_ci\_path);
592. $\_ci\_file = end($\_ci\_x);
593. }
594. else
595. {
596. //视图文件。
597. //下面两行操作也是为了让外部可以通过xxx.php或者直接xxx的方式进行传参而已。
598. $\_ci\_ext = pathinfo($\_ci\_view, PATHINFO\_EXTENSION);
599. $\_ci\_file = ($\_ci\_ext == '') ? $\_ci\_view.'.php' : $\_ci\_view;
601. foreach ($this->\_ci\_view\_paths as $view\_file => $cascade)
602. {
603. //从Loader::$\_ci\_view\_paths中遍历视图文件，如果找到则退出。
604. //（默认仅有APPPATH/view/下，当然也可以通过Loader::add\_package()方法设置）
605. if (file\_exists($view\_file.$\_ci\_file))
606. {
607. $\_ci\_path = $view\_file.$\_ci\_file;
608. $file\_exists = TRUE;
609. break;
610. }
612. //如果没有找到，会根据这个$cascade判断允不允许继续往下一个路径寻找视图文件。
613. if ( ! $cascade)
614. {
615. break;
616. }
617. }
618. }
620. //如果找不到文件（普通或视图都一样），则报错。
621. if ( ! $file\_exists && ! file\_exists($\_ci\_path))
622. {
623. show\_error('Unable to load the requested file: '.$\_ci\_file);
624. }
626. //下面这个也很关键，其实视图文件里面的代码都是在属于Loader组件的，什么意思？
627. //你可以随便写一个视图文件，然后在里面写上var\_dump($this);可以发现，这个$this，是指Loader。
628. //为什么会这样子呢？再往下面十几行代码的地方就说明了这一点。
629. //这里是把CI所有的属性都开放给Loader组件用，这样在视图文件里面就可以通过$this->xxx的方式调用控制器
630. //所有的东西。
632. $\_ci\_CI =& get\_instance();
633. foreach (get\_object\_vars($\_ci\_CI) as $\_ci\_key => $\_ci\_var)
634. {
635. if ( ! isset($this->$\_ci\_key))
636. {
637. $this->$\_ci\_key =& $\_ci\_CI->$\_ci\_key;
638. }
639. }
641. //在这里把在控制器里面通过$this->load->view("xxx",$data);中的$data解开，这就是为什么可以在视图文件
642. //中可以用$data里面的变量的原因。其实还可以通过Loader::vars()方法，设置这些变量，它们会首先保存在
643. //Loader::$\_ci\_cached\_vars中
644. if (is\_array($\_ci\_vars))
645. {
646. $this->\_ci\_cached\_vars = array\_merge($this->\_ci\_cached\_vars, $\_ci\_vars);
647. }
648. extract($this->\_ci\_cached\_vars);
650. //我们在控制器中调用$this->load->view()方法，实质视图并没有马上输出来，而是先将它放到缓冲区。
651. ob\_start();
653. //就是这个地方，下面if中有一句eval(xxxx)以及else中有include;而里面的xxxx正是我们要加载的视图文件，
654. //所以这就是为什么在视图文件里，var\_dump($this)，会告诉你当前这个$this是Loader组件，因为视图的代码都是相当于
655. //嵌入这个地方。
656. if ((bool) @ini\_get('short\_open\_tag') === FALSE AND config\_item('rewrite\_short\_tags') == TRUE)
657. {
658. echo eval('?>'.preg\_replace("/;\*\s\*\?>/", "; ?>", str\_replace('<?=', '<?php echo ', file\_get\_contents($\_ci\_path))));//'
659. }
660. else
661. {
662. include($\_ci\_path);
663. }
665. //经过上面的代码，我们的视图文件的内容已经放到了缓冲区了。
667. log\_message('debug', 'File loaded: '.$\_ci\_path);
669. //一般情况下，$\_ci\_return都为FLASE，即不要求通过$this->load->view()返回输出内容，而是直接放到缓冲区静候处理;
670. //当然你也可以先拿出数据，在控制器里面处理一下，再输出，例如在控制器中
671. //$output=$this->load->view("x",$data,TRUE);，当为TRUE的时候，下面的代码就起作用了。
672. if ($\_ci\_return === TRUE)
673. {
674. $buffer = ob\_get\_contents();
675. @ob\_end\_clean();
676. return $buffer;
677. }
679. //下面这个很关键，因为有可能当前这个视图文件是被另一个视图文件通过$this->view()方法引入，即视图文件嵌入视图文件
680. //从而导致多了一层缓冲。
681. //为了保证缓冲内容最后交给Output处理时，缓冲级别只比Loader组件加载时多1（这个1就是最父层的视图文件引起的）
682. //这里必须先flush掉当前层视图引起的这次缓冲，以保证Output正常工作。
683. if (ob\_get\_level() > $this->\_ci\_ob\_level + 1)
684. {
685. ob\_end\_flush();
686. }
687. else
688. {
689. //如果不是多1，则说明当前引入的视图文件就是直接在控制器里面引入的那个，而不是由某个视图文件再引入的。
691. //把缓冲区的内容交给Output组件并清空关闭缓冲区。
692. $\_ci\_CI->output->append\_output(ob\_get\_contents());
693. @ob\_end\_clean();
694. }
695. }
697. // --------------------------------------------------------------------
699. /\*\*
700. \* Load class
701. \*/
702. protected function \_ci\_load\_class($class, $params = NULL, $object\_name = NULL)
703. {
704. //去掉后缀.php，是为了方便外部可以通过xxx.php也可以通过xxx.php来传入类名。同时去掉两端的/。
705. $class = str\_replace('.php', '', trim($class, '/'));
707. //因为CI允许通过"dir1/dir2/classname"的格式来组织和加载类，所以还要判断类名中是否包括这些目录信息。
708. $subdir = '';
709. if (($last\_slash = strrpos($class, '/')) !== FALSE)
710. {
711. //目录部分
712. $subdir = substr($class, 0, $last\_slash + 1);
714. //类名部分
715. $class = substr($class, $last\_slash + 1);
716. }
718. //CI允许类文件以大写字母开头或者全小写，下面的遍历，就是在遍历这两种情况。
719. foreach (array(ucfirst($class), strtolower($class)) as $class)
720. {
721. $subclass = APPPATH.'libraries/'.$subdir.config\_item('subclass\_prefix').$class.'.php';
723. //是否有我们开发人员自己写的扩展当前类的扩展？如果有的话，把它加载进来。
724. if (file\_exists($subclass))
725. {
726. //先加载父类。
727. $baseclass = BASEPATH.'libraries/'.ucfirst($class).'.php';
729. if ( ! file\_exists($baseclass))
730. {
731. log\_message('error', "Unable to load the requested class: ".$class);
732. show\_error("Unable to load the requested class: ".$class);
733. }
735. if (in\_array($subclass, $this->\_ci\_loaded\_files))
736. {
738. if ( ! is\_null($object\_name))
739. {
740. $CI =& get\_instance();
741. //我们加载的类最终都是加载给超级控制器的，如果超级控制器已经有的话，那么我们没必要加载。
742. //如果没有，则实例它并加载给控制器。，
743. if ( ! isset($CI->$object\_name))
744. {
745. return $this->\_ci\_init\_class($class, config\_item('subclass\_prefix'), $params, $object\_name);
746. }
747. }
749. $is\_duplicate = TRUE;
750. log\_message('debug', $class." class already loaded. Second attempt ignored.");
751. return;
752. }
754. //加载类。
755. include\_once($baseclass);
756. include\_once($subclass);
757. //把已加载的类记录到Loader::\_ci\_loaded\_files中。
758. $this->\_ci\_loaded\_files[] = $subclass;
760. //调用Loader::\_ci\_init\_class()方法进而实例化。
761. return $this->\_ci\_init\_class($class, config\_item('subclass\_prefix'), $params, $object\_name);
762. }
764. //如果是没有写扩展。方法和上面大致相同，最后都是通过调用Loader::\_ci\_init\_class()方法进而实例化。
766. $is\_duplicate = FALSE;
767. foreach ($this->\_ci\_library\_paths as $path)
768. {
769. $filepath = $path.'libraries/'.$subdir.$class.'.php';
770. if ( ! file\_exists($filepath))
771. {
772. continue;
773. }
775. if (in\_array($filepath, $this->\_ci\_loaded\_files))
776. {
777. if ( ! is\_null($object\_name))
778. {
779. $CI =& get\_instance();
780. if ( ! isset($CI->$object\_name))
781. {
782. return $this->\_ci\_init\_class($class, '', $params, $object\_name);
783. }
784. }
786. $is\_duplicate = TRUE;
787. log\_message('debug', $class." class already loaded. Second attempt ignored.");
788. return;
789. }
791. include\_once($filepath);
792. $this->\_ci\_loaded\_files[] = $filepath;
793. return $this->\_ci\_init\_class($class, '', $params, $object\_name);
794. }
796. } // END FOREACH
798. //其实正常的话，上面如果找到此类就找到，没找到就没有了。不过CI在会这里做最后的尝试。会不会是放到了一个同名的
799. //子目录下。
800. if ($subdir == '')
801. {
802. $path = strtolower($class).'/'.$class;
803. return $this->\_ci\_load\_class($path, $params);
804. }
806. //没有找到就报错咯。
807. if ($is\_duplicate == FALSE)
808. {
809. log\_message('error', "Unable to load the requested class: ".$class);
810. show\_error("Unable to load the requested class: ".$class);
811. }
812. }
814. // --------------------------------------------------------------------
816. //此方法是用于实例化已经把类文件include进来的类。
817. protected function \_ci\_init\_class($class, $prefix = '', $config = FALSE, $object\_name = NULL)
818. {
819. // Is there an associated config file for this class? Note: these should always be lowercase
820. if ($config === NULL)
821. {
822. // Fetch the config paths containing any package paths
823. $config\_component = $this->\_ci\_get\_component('config');
825. if (is\_array($config\_component->\_config\_paths))
826. {
827. // Break on the first found file, thus package files
828. // are not overridden by default paths
829. foreach ($config\_component->\_config\_paths as $path)
830. {
831. // We test for both uppercase and lowercase, for servers that
832. // are case-sensitive with regard to file names. Check for environment
833. // first, global next
834. if (defined('ENVIRONMENT') AND file\_exists($path .'config/'.ENVIRONMENT.'/'.strtolower($class).'.php'))
835. {
836. include($path .'config/'.ENVIRONMENT.'/'.strtolower($class).'.php');
837. break;
838. }
839. elseif (defined('ENVIRONMENT') AND file\_exists($path .'config/'.ENVIRONMENT.'/'.ucfirst(strtolower($class)).'.php'))
840. {
841. include($path .'config/'.ENVIRONMENT.'/'.ucfirst(strtolower($class)).'.php');
842. break;
843. }
844. elseif (file\_exists($path .'config/'.strtolower($class).'.php'))
845. {
846. include($path .'config/'.strtolower($class).'.php');
847. break;
848. }
849. elseif (file\_exists($path .'config/'.ucfirst(strtolower($class)).'.php'))
850. {
851. include($path .'config/'.ucfirst(strtolower($class)).'.php');
852. break;
853. }
854. }
855. }
856. }
858. if ($prefix == '')
859. {
860. if (class\_exists('CI\_'.$class))
861. {
862. $name = 'CI\_'.$class;
863. }
864. elseif (class\_exists(config\_item('subclass\_prefix').$class))
865. {
866. $name = config\_item('subclass\_prefix').$class;
867. }
868. else
869. {
870. $name = $class;
871. }
872. }
873. else
874. {
875. $name = $prefix.$class;
876. }
878. // Is the class name valid?
879. if ( ! class\_exists($name))
880. {
881. log\_message('error', "Non-existent class: ".$name);
882. show\_error("Non-existent class: ".$class);
883. }
885. // Set the variable name we will assign the class to
886. // Was a custom class name supplied? If so we'll use it
887. $class = strtolower($class);
889. if (is\_null($object\_name))
890. {
891. $classvar = ( ! isset($this->\_ci\_varmap[$class])) ? $class : $this->\_ci\_varmap[$class];
892. }
893. else
894. {
895. $classvar = $object\_name;
896. }
898. // Save the class name and object name
899. $this->\_ci\_classes[$class] = $classvar;
901. // Instantiate the class
902. $CI =& get\_instance();
903. if ($config !== NULL)
904. {
905. $CI->$classvar = new $name($config);
906. }
907. else
908. {
909. $CI->$classvar = new $name;
910. }
911. }
913. // --------------------------------------------------------------------
915. /\*\*
916. \* Autoloader
917. \*
918. \* The config/autoload.php file contains an array that permits sub-systems,
919. \* libraries, and helpers to be loaded automatically.
920. \*
921. \*/
922. private function \_ci\_autoloader()
923. {
924. if (defined('ENVIRONMENT') AND file\_exists(APPPATH.'config/'.ENVIRONMENT.'/autoload.php'))
925. {
926. include(APPPATH.'config/'.ENVIRONMENT.'/autoload.php');
927. }
928. else
929. {
930. include(APPPATH.'config/autoload.php');
931. }
933. if ( ! isset($autoload))
934. {
935. return FALSE;
936. }
938. // Autoload packages
939. if (isset($autoload['packages']))
940. {
941. foreach ($autoload['packages'] as $package\_path)
942. {
943. $this->add\_package\_path($package\_path);
944. }
945. }
947. // Load any custom config file
948. if (count($autoload['config']) > 0)
949. {
950. $CI =& get\_instance();
951. foreach ($autoload['config'] as $key => $val)
952. {
953. $CI->config->load($val);
954. }
955. }
957. // Autoload helpers and languages
958. foreach (array('helper', 'language') as $type)
959. {
960. if (isset($autoload[$type]) AND count($autoload[$type]) > 0)
961. {
962. $this->$type($autoload[$type]);
963. }
964. }
966. // A little tweak to remain backward compatible
967. // The $autoload['core'] item was deprecated
968. if ( ! isset($autoload['libraries']) AND isset($autoload['core']))
969. {
970. $autoload['libraries'] = $autoload['core'];
971. }
973. // Load libraries
974. if (isset($autoload['libraries']) AND count($autoload['libraries']) > 0)
975. {
976. // Load the database driver.
977. if (in\_array('database', $autoload['libraries']))
978. {
979. $this->database();
980. $autoload['libraries'] = array\_diff($autoload['libraries'], array('database'));
981. }
983. // Load all other libraries
984. foreach ($autoload['libraries'] as $item)
985. {
986. $this->library($item);
987. }
988. }
990. // Autoload models
991. if (isset($autoload['model']))
992. {
993. $this->model($autoload['model']);
994. }
995. }
997. // --------------------------------------------------------------------
999. /\*\*
1000. \* Object to Array
1001. \*
1002. \* Takes an object as input and converts the class variables to array key/vals
1003. \*
1004. \*/
1005. protected function \_ci\_object\_to\_array($object)
1006. {
1007. return (is\_object($object)) ? get\_object\_vars($object) : $object;
1008. }
1010. // --------------------------------------------------------------------
1012. /\*\*
1013. \* Get a reference to a specific library or model
1014. \*
1015. \*/
1016. protected function &\_ci\_get\_component($component)
1017. {
1018. $CI =& get\_instance();
1019. return $CI->$component;
1020. }
1022. // --------------------------------------------------------------------
1024. /\*\*
1025. \* Prep filename
1026. \*
1027. \* This function preps the name of various items to make loading them more reliable.
1028. \*
1029. \*/
1030. protected function \_ci\_prep\_filename($filename, $extension)
1031. {
1032. if ( ! is\_array($filename))
1033. {
1034. return array(strtolower(str\_replace('.php', '', str\_replace($extension, '', $filename)).$extension));
1035. }
1036. else
1037. {
1038. foreach ($filename as $key => $val)
1039. {
1040. $filename[$key] = strtolower(str\_replace('.php', '', str\_replace($extension, '', $val)).$extension);
1041. }
1043. return $filename;
1044. }
1045. }
1046. }

[CodeIgniter源码分析之Output.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Output.php%20–%20Calix的博客_files\CodeIgniter源码分析之Output.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
3. // ------------------------------------------------------------------------
5. /\*\*
6. \* Output Class
7. \*
8. \* Output组件其实有很多有用的方法，不过一般情况下，你不会直接去用到它们。
9. \* 这里主要以Output::\_display\_cache()和Output::\_display()为两条主线来探究。
10. \*/
11. class CI\_Output {
13. /\*\*
14. \* Current output string
15. \*/
16. protected $final\_output;
18. /\*\*
19. \* Cache expiration time
20. \*/
21. protected $cache\_expiration = 0;
23. /\*\*
24. \* List of server headers
25. \*/
26. protected $headers = array();
28. /\*\*
29. \* List of mime types
30. \*/
31. protected $mime\_types = array();
33. /\*\*
34. \* Determines wether profiler is enabled
35. \*/
36. protected $enable\_profiler = FALSE;
38. /\*\*
39. \* Determines if output compression is enabled
40. \* @access protected
41. \*/
42. protected $\_zlib\_oc = FALSE;
44. /\*\*
45. \* List of profiler sections
46. \*/
47. protected $\_profiler\_sections = array();
49. /\*\*
50. \* Whether or not to parse variables like {elapsed\_time} and {memory\_usage}
51. \*/
52. protected $parse\_exec\_vars = TRUE;
54. /\*\*
55. \* Constructor
56. \*/
57. function \_\_construct()
58. {
59. $this->\_zlib\_oc = @ini\_get('zlib.output\_compression');
61. if (defined('ENVIRONMENT') AND file\_exists(APPPATH.'config/'.ENVIRONMENT.'/mimes.php'))
62. {
63. include APPPATH.'config/'.ENVIRONMENT.'/mimes.php';
64. }
65. else
66. {
67. include APPPATH.'config/mimes.php';
68. }
70. $this->mime\_types = $mimes;
72. log\_message('debug', "Output Class Initialized");
73. }
75. // --------------------------------------------------------------------
77. /\*\*
78. \* Get Output
79. \*/
80. function get\_output()
81. {
82. //返回最终输出。
83. return $this->final\_output;
84. }
86. // --------------------------------------------------------------------
88. /\*\*
89. \* Set Output
90. \*/
91. function set\_output($output)
92. {
93. //设置最终输出。
94. $this->final\_output = $output;
96. return $this;
97. }
99. // --------------------------------------------------------------------
101. /\*\*
102. \* Append Output
103. \*/
104. //通过此方法给Output::$final\_output加上输出内容，最终这些内容会在Output::\_display()中被输出。
105. //此方法在Loader.php中被调用。详见Loader.php中的Loader::view()方法及Loader::\_ci\_load()方法。
106. function append\_output($output)
107. {
108. if ($this->final\_output == '')
109. {
110. $this->final\_output = $output;
111. }
112. else
113. {
114. $this->final\_output .= $output;
115. }
117. return $this;
118. }
120. // --------------------------------------------------------------------
122. /\*\*
123. \* Set Header
124. \*/
125. /\*\*
126. \* 接下来的三个方法都是设置页面内容输出前的头信息。
127. \*/
128. function set\_header($header, $replace = TRUE)
129. {
130. if ($this->\_zlib\_oc && strncasecmp($header, 'content-length', 14) == 0)
131. {
132. return;
133. }
135. $this->headers[] = array($header, $replace);
137. return $this;
138. }
140. // --------------------------------------------------------------------
142. /\*\*
143. \* Set Content Type Header
144. \*/
145. function set\_content\_type($mime\_type)
146. {
147. if (strpos($mime\_type, '/') === FALSE)
148. {
149. $extension = ltrim($mime\_type, '.');
151. if (isset($this->mime\_types[$extension]))
152. {
153. $mime\_type =& $this->mime\_types[$extension];
155. if (is\_array($mime\_type))
156. {
157. $mime\_type = current($mime\_type);
158. }
159. }
160. }
162. $header = 'Content-Type: '.$mime\_type;
164. $this->headers[] = array($header, TRUE);
166. return $this;
167. }
169. // --------------------------------------------------------------------
171. /\*\*
172. \* Set HTTP Status Header
173. \*/
174. function set\_status\_header($code = 200, $text = '')
175. {
176. set\_status\_header($code, $text);
178. return $this;
179. }
181. // --------------------------------------------------------------------
183. /\*\*
184. \* Enable/disable Profiler
185. \*/
186. //是否开启评测器。
187. function enable\_profiler($val = TRUE)
188. {
189. $this->enable\_profiler = (is\_bool($val)) ? $val : TRUE;
191. return $this;
192. }
194. // --------------------------------------------------------------------
196. /\*\*
197. \* Set Profiler Sections
198. \*/
199. function set\_profiler\_sections($sections)
200. {
201. foreach ($sections as $section => $enable)
202. {
203. $this->\_profiler\_sections[$section] = ($enable !== FALSE) ? TRUE : FALSE;
204. }
206. return $this;
207. }
209. // --------------------------------------------------------------------
211. /\*\*
212. \* Set Cache
213. \*/
214. function cache($time)
215. {
216. //设置缓存时长。
217. $this->cache\_expiration = ( ! is\_numeric($time)) ? 0 : $time;
219. return $this;
220. }
222. // --------------------------------------------------------------------
224. /\*\*
225. \* Display Output
226. \*/
227. function \_display($output = '')
228. {
229. //为什么这里要用global来调用这两个组件呢？不是可以通过$CI这个超级控制器来调用？
230. //其实就是因为这个\_display方法，被调用的方式有两种，其中有可能是在CodeIgniter.php中调用
231. //Output::\_display\_cache();的时候间接被调用了，而此时& get\_instance()这个方法压根还没
232. //被定义。详见CodeIgniter.php中代码定义和调用的顺序。
233. global $BM, $CFG;
235. //当然如果可以拿到超级控制器，我们先拿过来。
236. if (class\_exists('CI\_Controller'))
237. {
238. $CI =& get\_instance();
239. }
241. // --------------------------------------------------------------------
243. //如果$output为空，其实往往这是非缓存方式调用的时候。我们将使用Output::final\_output。（如果是正常流程的输出
244. //方式，而不是缓存的话，这个属性其实在Loader::view()的时候调用Output::append\_output()获得输出内容。）
245. if ($output == '')
246. {
247. $output =& $this->final\_output;
248. }
250. // --------------------------------------------------------------------
252. //Output::$cache\_expiration其实就是缓存时长，就是平时我们在控制器里面$this->output->cache(n)设置的时长
253. //现实手段就是使这个Output::$cache\_expiration有一定的值，然后程序执行到这里时根据此值判断是否要缓存，
254. //如果要缓存就生成缓存文件。（注意如果是\_display\_cache间接调用的话，$this->cache\_expiraton是一定为0的，因为
255. //没有经历过在控制器中调用$this->output->cache(n)。）
256. if ($this->cache\_expiration > 0 && isset($CI) && ! method\_exists($CI, '\_output'))
257. {
258. //上面有个判断$CI是否有\_output方法，其实是提供一个机会让我们自定义处理输出。
259. //生成缓存文件。
260. $this->\_write\_cache($output);
261. }
263. // --------------------------------------------------------------------
265. $elapsed = $BM->elapsed\_time('total\_execution\_time\_start', 'total\_execution\_time\_end');
267. if ($this->parse\_exec\_vars === TRUE)
268. {
269. //系统的总体运行时间和内存消耗就是在这里替换的。呵呵。上面的Output::$parse\_exec\_vars就是设置要不要替换。
270. $memory = ( ! function\_exists('memory\_get\_usage')) ? '0' : round(memory\_get\_usage()/1024/1024, 2).'MB';
272. $output = str\_replace('{elapsed\_time}', $elapsed, $output);
273. $output = str\_replace('{memory\_usage}', $memory, $output);
274. }
276. // --------------------------------------------------------------------
277. //压缩传输的处理。
278. if ($CFG->item('compress\_output') === TRUE && $this->\_zlib\_oc == FALSE)
279. {
280. if (extension\_loaded('zlib'))
281. {
282. if (isset($\_SERVER['HTTP\_ACCEPT\_ENCODING']) AND strpos($\_SERVER['HTTP\_ACCEPT\_ENCODING'], 'gzip') !== FALSE)
283. {
284. ob\_start('ob\_gzhandler');
285. }
286. }
287. }
289. // --------------------------------------------------------------------
290. if (count($this->headers) > 0)
291. {
292. foreach ($this->headers as $header)
293. {
294. @header($header[0], $header[1]);
295. }
296. }
298. // --------------------------------------------------------------------
299. //如果没有超级控制器，可以证明当前是在处理一个缓存的输出。不过利用这个方式来判断，真的有点那个。。。
300. if ( ! isset($CI))
301. {
302. echo $output;//输出缓存内容。结束本函数。
303. log\_message('debug', "Final output sent to browser");
304. log\_message('debug', "Total execution time: ".$elapsed);
305. return TRUE;
306. }
308. // --------------------------------------------------------------------
309. //这里是一个评测器，如果有开启就调用，会生成一些报告到页面尾部用于辅助我们调试。我用CI的时候其实没有开启过，厄。
310. if ($this->enable\_profiler == TRUE)
311. {
312. $CI->load->library('profiler');
314. if ( ! empty($this->\_profiler\_sections))
315. {
316. $CI->profiler->set\_sections($this->\_profiler\_sections);
317. }
319. if (preg\_match("|</body>.\*?</html>|is", $output))
320. {
321. $output = preg\_replace("|</body>.\*?</html>|is", '', $output);
322. $output .= $CI->profiler->run();
323. $output .= '</body></html>';
324. }
325. else
326. {
327. $output .= $CI->profiler->run();
328. }
329. }
331. // --------------------------------------------------------------------
333. //如果我们有在当前的控制器里面定义了\_output这个方法，那么可以利用这个输出做你想做的东西，这个也是很不错的功能。
334. if (method\_exists($CI, '\_output'))
335. {
336. $CI->\_output($output);
337. }
338. else
339. {
340. //如果没有定义\_output，就默认简单输出。
341. echo $output; // Send it to the browser!
342. }
344. log\_message('debug', "Final output sent to browser");
345. log\_message('debug', "Total execution time: ".$elapsed);
346. }
348. // --------------------------------------------------------------------
350. /\*\*
351. \* Write a Cache File
352. \*/
353. /\*\*
354. \* 写入缓存。在Output::\_display()中，判断需要缓存页面时，则调用此方法写入缓存。
355. \*/
356. function \_write\_cache($output)
357. {
358. //下面很多操作其实和读缓存的时候差不多。
360. $CI =& get\_instance();
361. $path = $CI->config->item('cache\_path');
363. $cache\_path = ($path == '') ? APPPATH.'cache/' : $path;
365. if ( ! is\_dir($cache\_path) OR ! is\_really\_writable($cache\_path))
366. {
367. log\_message('error', "Unable to write cache file: ".$cache\_path);
368. return;
369. }
371. $uri = $CI->config->item('base\_url').
372. $CI->config->item('index\_page').
373. $CI->uri->uri\_string();
375. $cache\_path .= md5($uri);
377. if ( ! $fp = @fopen($cache\_path, FOPEN\_WRITE\_CREATE\_DESTRUCTIVE))
378. {
379. log\_message('error', "Unable to write cache file: ".$cache\_path);
380. return;
381. }
383. //计算缓存文件过期时间。
384. $expire = time() + ($this->cache\_expiration \* 60);
386. if (flock($fp, LOCK\_EX))
387. {
388. //按CI的格式写入缓存内容。
389. fwrite($fp, $expire.'TS--->'.$output);
390. flock($fp, LOCK\_UN);
391. }
392. else
393. {
394. log\_message('error', "Unable to secure a file lock for file at: ".$cache\_path);
395. return;
396. }
397. fclose($fp);
398. @chmod($cache\_path, FILE\_WRITE\_MODE);
400. log\_message('debug', "Cache file written: ".$cache\_path);
401. }
403. // --------------------------------------------------------------------
405. /\*\*
406. \* Update/serve a cached file
407. \*/
408. //在CodeIgniter.php里面有调用此方法，此方法是负责缓存的输出，如果在CodeIgniter.php中调用此方法有输出，则
409. //本次请求的运行将直接结束，直接以缓存输出作为响应。
410. function \_display\_cache(&$CFG, &$URI)
411. {
412. //取得保存缓存的路径
413. $cache\_path = ($CFG->item('cache\_path') == '') ? APPPATH.'cache/' : $CFG->item('cache\_path');
415. //一条准确的路由都会对应一个缓存文件，缓存文件是对应路由字符串的md5密文。
416. $uri = $CFG->item('base\_url').
417. $CFG->item('index\_page').
418. $URI->uri\_string;
420. //计算出当前请求对应缓存文件的完整文件路径。
421. $filepath = $cache\_path.md5($uri);
423. //如果没有此缓存文件，获取缓存内容失败，则可以返回FALSE。
424. if ( ! @file\_exists($filepath))
425. {
426. return FALSE;
427. }
429. //如果有此缓存文件，但是无法读，获取缓存内容失败，同样返回FALSE。
430. if ( ! $fp = @fopen($filepath, FOPEN\_READ))
431. {
432. return FALSE;
433. }
435. //打开到缓存文件，并以$fp作为句柄。下一步先取得共享锁（读取）。
436. flock($fp, LOCK\_SH);
438. //读cache并保存到$cache中
439. $cache = '';
440. if (filesize($filepath) > 0)
441. {
442. $cache = fread($fp, filesize($filepath));
443. }
445. //解锁
446. flock($fp, LOCK\_UN);
447. //关闭文件连接。
448. fclose($fp);
450. //下面这个TS--->字样，只是因为CI的缓存文件里面的内容是规定以数字＋TS--->开头而已。这个数字是代表创建时间。
451. //如果不符合此结构，可视为非CI的缓存文件，或者文件已损坏，获取缓存内容失败，返回FALSE。
452. //如果匹配成功，则$match[1]中保存的是"1346901048TS--->"字样。其实在CI的这个版本$match[0]保存的是和
453. //$match[1]相同的内容，为什么要分开？我觉得是为了以后扩展和方便改动吧，理解成
454. //$match[0]是除页面内容之外的附加信息。
455. //$match[1]是附加信息中和时间有关的信息。
456. if ( ! preg\_match("/(\d+TS--->)/", $cache, $match))
457. {
458. return FALSE;
459. }
461. //去掉TS--->后，利用剩下的数字判断缓存是否已经过期，如果过期了，就把它删除，同样视为获取缓存内容失败（过期），
462. //返回FALSE
463. if (time() >= trim(str\_replace('TS--->', '', $match['1'])))
464. {
465. if (is\_really\_writable($cache\_path))
466. {
467. @unlink($filepath);
468. log\_message('debug', "Cache file has expired. File deleted");
469. return FALSE;
470. }
471. }
473. //来到这里，说明了能够顺利获得缓存，则去掉附加信息（$match[0]）后，调用Output::\_display()方法输出缓存。并返回
474. $this->\_display(str\_replace($match['0'], '', $cache));
475. log\_message('debug', "Cache file is current. Sending it to browser.");
476. return TRUE;
477. }
479. }

[CodeIgniter源码分析之Hook.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Hook.php%20–%20Calix的博客_files\CodeIgniter源码分析之Hook.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. // ------------------------------------------------------------------------
4. /\*\*
5. \* CodeIgniter Hooks Class
6. \*/
7. class CI\_Hooks {
9. /\*\*
10. \* Determines wether hooks are enabled
11. \*/
12. var $enabled = FALSE;
14. /\*\*
15. \* List of all hooks set in config/hooks.php
16. \*/
17. var $hooks = array();
19. /\*\*
20. \* Determines wether hook is in progress, used to prevent infinte loops
21. \*/
22. var $in\_progress = FALSE;
24. /\*\*
25. \* Constructor
26. \*/
27. function \_\_construct()
28. {
29. $this->\_initialize();
30. log\_message('debug', "Hooks Class Initialized");
31. }
33. // --------------------------------------------------------------------
35. /\*\*
36. \* Initialize the Hooks Preferences
37. \*/
38. function \_initialize()
39. {
40. $CFG =& load\_class('Config', 'core');
42. //如果配置文件中设置了是不允许hooks，则直接返回退出本函数。
43. if ($CFG->item('enable\_hooks') == FALSE)
44. {
45. return;
46. }
48. //要使用到的钩子，必须在配置目录下的hooks.php里面定义好。否则无法使用。
49. if (defined('ENVIRONMENT') AND is\_file(APPPATH.'config/'.ENVIRONMENT.'/hooks.php'))
50. {
51. include(APPPATH.'config/'.ENVIRONMENT.'/hooks.php');
52. }
53. elseif (is\_file(APPPATH.'config/hooks.php'))
54. {
55. include(APPPATH.'config/hooks.php');
56. }
58. if ( ! isset($hook) OR ! is\_array($hook))
59. {
60. return;
61. }
63. //把钩子信息都保存到Hook组件中。
64. $this->hooks =& $hook;
65. $this->enabled = TRUE;
66. }
68. // --------------------------------------------------------------------
70. /\*\*
71. \* Call Hook
72. \* 外部其实就是调用这个\_call\_hook函数进行调用钩子程序。而此方法中再调用\_run\_hook去执行相应的钩子。
73. \*/
74. function \_call\_hook($which = '')
75. {
76. if ( ! $this->enabled OR ! isset($this->hooks[$which]))
77. {
78. return FALSE;
79. }
81. //同一个位置可以执行多个hook
82. if (isset($this->hooks[$which][0]) AND is\_array($this->hooks[$which][0]))
83. {
84. foreach ($this->hooks[$which] as $val)
85. {
86. $this->\_run\_hook($val);
87. }
88. }
89. else
90. {
91. $this->\_run\_hook($this->hooks[$which]);
92. }
94. return TRUE;
95. }
97. // --------------------------------------------------------------------
99. /\*\*
100. \* Run Hook
101. \*/
102. function \_run\_hook($data)
103. {
104. //一般来说，这个$data会有：类名，方法名，参数，类文件路径等参数。
105. if ( ! is\_array($data))
106. {
107. return FALSE;
108. }
110. // -----------------------------------
111. // Safety - Prevents run-away loops
112. // -----------------------------------
114. //如果调用某一个hook，执行某些脚本，而有可能这些脚本里面再会触发其它hook，如果这个其它hook里面又包含了当前
115. //的hook，那么就会进入死循环，这个in\_progress的存在就是阻止这种情况。
116. if ($this->in\_progress == TRUE)
117. {
118. return;
119. }
121. //下面都是一些执行钩子的预处理，包括判断类文件是否存在，类和方法是否正确等等。
122. // -----------------------------------
123. // Set file path
124. // -----------------------------------
126. if ( ! isset($data['filepath']) OR ! isset($data['filename']))
127. {
128. return FALSE;
129. }
131. $filepath = APPPATH.$data['filepath'].'/'.$data['filename'];
133. if ( ! file\_exists($filepath))
134. {
135. return FALSE;
136. }
138. // -----------------------------------
139. // Set class/function name
140. // -----------------------------------
142. $class = FALSE;
143. $function = FALSE;
144. $params = '';
146. if (isset($data['class']) AND $data['class'] != '')
147. {
148. $class = $data['class'];
149. }
151. if (isset($data['function']))
152. {
153. $function = $data['function'];
154. }
156. if (isset($data['params']))
157. {
158. $params = $data['params'];
159. }
161. if ($class === FALSE AND $function === FALSE)
162. {
163. return FALSE;
164. }
166. // -----------------------------------
167. // Set the in\_progress flag
168. // -----------------------------------
170. //在开始执行钩子相应的程序之前，先把当前hook的状态设为正在运行中。
171. $this->in\_progress = TRUE;
173. // -----------------------------------
174. // Call the requested class and/or function
175. // -----------------------------------
177. //执行
178. if ($class !== FALSE)
179. {
180. if ( ! class\_exists($class))
181. {
182. require($filepath);
183. }
185. $HOOK = new $class;
186. $HOOK->$function($params);
187. }
188. else
189. {
190. if ( ! function\_exists($function))
191. {
192. require($filepath);
193. }
195. $function($params);
196. }
198. //执行相应程序完毕后，重新把当前hook的状态改为非运行中，以让它可以再次被触发。
199. $this->in\_progress = FALSE;
200. return TRUE;
201. }
203. }

[CodeIgniter源码分析之Config.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Config.php%20–%20Calix的博客_files\CodeIgniter源码分析之Config.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
3. // ------------------------------------------------------------------------
5. /\*\*
6. \* CodeIgniter Config Class
7. \*/
8. class CI\_Config {
10. /\*\*
11. \* List of all loaded config values
12. \*/
13. var $config = array();
15. /\*\*
16. \* List of all loaded config files
17. \*/
18. var $is\_loaded = array();
20. /\*\*
21. \* List of paths to search when trying to load a config file
22. \*/
23. var $\_config\_paths = array(APPPATH);
25. /\*\*
26. \* Constructor
27. \*/
28. function \_\_construct()
29. {
30. $this->config =& get\_config();
31. log\_message('debug', "Config Class Initialized");
33. //在config/config.php里面有个配置项是base\_url，它并不是必须配置项，如果没有配置，则系统就在这个地方
34. //自己去它进行赋值。
35. if ($this->config['base\_url'] == '')
36. {
37. //一般来说，如果通过http访问网站的话，这个值都会有的。
38. if (isset($\_SERVER['HTTP\_HOST']))
39. {
40. //判断是否通过https方式访问。
41. $base\_url = isset($\_SERVER['HTTPS']) && strtolower($\_SERVER['HTTPS']) !== 'off' ? 'https' : 'http';
42. $base\_url .= '://'. $\_SERVER['HTTP\_HOST'];
43. //去掉文件名部分。
44. $base\_url .= str\_replace(basename($\_SERVER['SCRIPT\_NAME']), '', $\_SERVER['SCRIPT\_NAME']);
45. }
47. else
48. {
49. //如果发现没有$\_SERVER['HTTP\_HOST']，则直接设置为localhost
50. $base\_url = 'http://localhost/';
51. }
53. //保存到base\_url中，以后像辅助函数uri\_helper就可以通过base\_url()调用出Config组件此值。
54. $this->set\_item('base\_url', $base\_url);
55. }
56. }
58. // --------------------------------------------------------------------
60. /\*\*
61. \* Load Config File
62. \* 先解释一下load方法的参数，$file就是配置文件名。配置文件目录一般为应用目录(application)/config/下
63. \* 下面会有很多个针对不同方面配置的文件，而我们通过Config组件加载的配置信息都会保存在Config::$config这个
64. \* 属性里面，所以第二个参数$use\_sections就是设置是否当前配置文件是否以独立一个数组的形式充当Config::$config
65. \* 的一个元素加入，如果为true，则$config是一个两层的数组，如果为false，则单纯将配置文件里面的配置信息合并。
66. \* 例如配置文件abc.php，如果为true，则会以$config['abc']['xxx']的形式保存，否则直接合并即会有
67. \* $config['xxx']。
68. \* 第三个参数只是设置要不要报错而已，如果为true，则只会返回false，如果为false则直接在函数执行时报错。
69. \*/
70. function load($file = '', $use\_sections = FALSE, $fail\_gracefully = FALSE)
71. {
72. //接下来这一行代码是为了方便我们调用的时候既可以以xxx.php的形式传参，也可以只以xxx（无后缀）的形式。
73. //另外如果$file为空，则默认是加载config.php
74. $file = ($file == '') ? 'config' : str\_replace('.php', '', $file);
76. $found = FALSE;
77. $loaded = FALSE;
79. //这个$this->\_config\_paths默认只有应用目录application/
80. foreach ($this->\_config\_paths as $path)
81. {
82. //分别从某特定环境的配置目录和默认的配置目录里面寻找。
83. $check\_locations = defined('ENVIRONMENT')
84. ? array(ENVIRONMENT.'/'.$file, $file)
85. : array($file);
87. foreach ($check\_locations as $location)
88. {
89. $file\_path = $path.'config/'.$location.'.php';
91. if (in\_array($file\_path, $this->is\_loaded, TRUE))
92. {
93. $loaded = TRUE;
94. continue 2;//如果是已经加载过了，那么在Config::$config里面理应当有，所以直接跳出最外层循环。
95. }
97. if (file\_exists($file\_path))
98. {
99. $found = TRUE;//如果找到了一个，就不再找了。所以相同的配置文件仅会有一个有效。
100. break;
101. }
102. }
104. //$found是用于判断在此$path里面，遍历上面的$check\_locations有没有找到
105. //而$load则是用于判断两层遍历以后，最终有没有把配置文件加载进来。
106. if ($found === FALSE)
107. {
108. continue;
109. }
111. //配置文件就是在这个地方加载的，
112. include($file\_path);
114. //下面这句可以看出，我们在包含的配置文件里面必须要有名为$config的数组。
115. //如果配置信息格式不合法，看情况（$$fail\_gracefully的作用）处理错误。
116. if ( ! isset($config) OR ! is\_array($config))
117. {
119. if ($fail\_gracefully === TRUE)
120. {
121. return FALSE;
122. }
123. show\_error('Your '.$file\_path.' file does not appear to contain a valid configuration array.');
124. }
126. //下面就是$use\_sections的作用，根据它来规定当前加载的配置信息的保存形式。
127. if ($use\_sections === TRUE)
128. {
129. if (isset($this->config[$file]))
130. {
131. $this->config[$file] = array\_merge($this->config[$file], $config);
132. }
133. else
134. {
135. $this->config[$file] = $config;
136. }
137. }
138. else
139. {
140. $this->config = array\_merge($this->config, $config);
141. }
143. //保存哪些文件已经加载过，下次再调用此load方法的时候，通过它来避免重复加载，减少不必要的操作。
144. $this->is\_loaded[] = $file\_path;
145. unset($config);
147. $loaded = TRUE;
148. log\_message('debug', 'Config file loaded: '.$file\_path);
149. break;
150. }
152. //加载失败，按情况处理错误。
153. if ($loaded === FALSE)
154. {
155. if ($fail\_gracefully === TRUE)
156. {
157. return FALSE;
158. }
159. show\_error('The configuration file '.$file.'.php'.' does not exist.');
160. }
162. //来到这里，说明了一切都很顺利，返回true。
163. return TRUE;
164. }
166. // --------------------------------------------------------------------
168. /\*\*
169. \* Fetch a config file item
170. \* 取得某一配置项的内容，如果知道上面Config::load($file, $use\_sections, $fail\_gracefully);方法
171. \* 中$use\_sections的意义的话，那个下面的$index意义就很容易理解了。
172. \*/
173. function item($item, $index = '')
174. {
175. if ($index == '')
176. {
177. if ( ! isset($this->config[$item]))
178. {
179. return FALSE;
180. }
182. $pref = $this->config[$item];
183. }
184. else
185. {
186. if ( ! isset($this->config[$index]))
187. {
188. return FALSE;
189. }
191. if ( ! isset($this->config[$index][$item]))
192. {
193. return FALSE;
194. }
196. $pref = $this->config[$index][$item];
197. }
199. return $pref;
200. }
202. // --------------------------------------------------------------------
204. /\*\*
205. \* Fetch a config file item - adds slash after item (if item is not empty)
206. \*/
207. //此方法仅仅是对配置信息进行一些修剪处理而已。
208. function slash\_item($item)
209. {
210. if ( ! isset($this->config[$item]))
211. {
212. return FALSE;
213. }
214. //如果此配置项仅仅是包含一些对配置无效的字符，则直接返回空。
215. if( trim($this->config[$item]) == '')
216. {
217. return '';
218. }
220. //保证以一条/结尾。
221. return rtrim($this->config[$item], '/').'/';
222. }
224. // --------------------------------------------------------------------
226. /\*\*
227. \* Site URL
228. \*/
229. //我们经常通过url\_helper的site\_url获得我们在项目中想要的路径，其实真正执行的是Config::site\_url()这个方法。
230. function site\_url($uri = '')
231. {
232. //$uri参数实质可以是数组的
234. if ($uri == '')
235. {
236. return $this->slash\_item('base\_url').$this->item('index\_page');
237. }
239. //根据当前的路由格式返回相应的uri\_string
240. if ($this->item('enable\_query\_strings') == FALSE)
241. {
242. $suffix = ($this->item('url\_suffix') == FALSE) ? '' : $this->item('url\_suffix');
243. return $this->slash\_item('base\_url').$this->slash\_item('index\_page').$this->\_uri\_string($uri).$suffix;
244. }
245. else
246. {
247. return $this->slash\_item('base\_url').$this->item('index\_page').'?'.$this->\_uri\_string($uri);
248. }
249. }
251. // -------------------------------------------------------------
253. /\*\*
254. \* Base URL
255. \*/
256. function base\_url($uri = '')
257. {
258. return $this->slash\_item('base\_url').ltrim($this->\_uri\_string($uri),'/');
259. }
261. // -------------------------------------------------------------
263. /\*\*
264. \* Build URI string for use in Config::site\_url() and Config::base\_url()
265. \*/
266. protected function \_uri\_string($uri)
267. {
268. /\*\*
269. \* 按当前规定路由格式，返回正确的uri\_string.
270. \* 主要是如果当参数$uri是数组的时候的一些处理。
271. \*/
272. if ($this->item('enable\_query\_strings') == FALSE)
273. {
274. if (is\_array($uri))
275. {
276. $uri = implode('/', $uri);
277. }
278. $uri = trim($uri, '/');
279. }
280. else
281. {
282. if (is\_array($uri))
283. {
284. $i = 0;
285. $str = '';
286. foreach ($uri as $key => $val)
287. {
288. $prefix = ($i == 0) ? '' : '&';
289. $str .= $prefix.$key.'='.$val;
290. $i++;
291. }
292. $uri = $str;
293. }
294. }
295. return $uri;
296. }
298. // --------------------------------------------------------------------
300. /\*\*
301. \* System URL
302. \*/
303. function system\_url()
304. {
305. //厄，下面这行这么奇葩的代码，其实只是为拿到系统目录的路径而已。
306. //正则部分是首先去掉BASEPATH中多余重复的“/”,然后再拆分为数组。最后通过end()函数来拿到系统目录名。
307. $x = explode("/", preg\_replace("|/\*(.+?)/\*$|", "\\1", BASEPATH));
308. return $this->slash\_item('base\_url').end($x).'/';
309. }
311. // --------------------------------------------------------------------
313. /\*\*
314. \* Set a config file item
315. \*/
316. function set\_item($item, $value)
317. {
318. $this->config[$item] = $value;
319. }
321. // --------------------------------------------------------------------
323. /\*\*
324. \* Assign to Config
325. \*/
326. /\*\*
327. \* 下面这个方法在CodeIgniter.php中调用过，是为把在index.php里设置的配置信息交给Config组件。
328. \* 实质也是通过上面的Config::set\_item();方法设置。
329. \*/
330. function \_assign\_to\_config($items = array())
331. {
332. if (is\_array($items))
333. {
334. foreach ($items as $key => $val)
335. {
336. $this->set\_item($key, $val);
337. }
338. }
339. }
340. }

[CodeIgniter源码分析之Benchmark.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Benchmark.php%20–%20Calix的博客_files\CodeIgniter源码分析之Benchmark.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. // ------------------------------------------------------------------------
4. /\*\*
5. \* CodeIgniter Benchmark Class
6. \*/
7. class CI\_Benchmark {
9. /\*\*
10. \* List of all benchmark markers and when they were added
11. \*/
12. var $marker = array();
14. // --------------------------------------------------------------------
16. /\*\*
17. \* Set a benchmark marker
18. \*/
19. function mark($name)
20. {
21. //这个方法其实很简单，就是在程序的任意地方调用地方法时，会记录当前的时间点。
22. $this->marker[$name] = microtime();
23. }
25. // --------------------------------------------------------------------
27. /\*\*
28. \* Calculates the time difference between two marked points.
29. \* 计算出两个时间点之间的时间。
30. \*/
31. function elapsed\_time($point1 = '', $point2 = '', $decimals = 4)
32. {
33. /\*
34. \* 如果没有给出明确的时间点，那么会计算出整个程序运行的时间。
35. \* 怎么可以做到计算出整个程序的运行时间的呢？其实执行此计算的是Output组件。
36. \* 而调用Benchmark::elapsed\_time();（无参数）的时候，实质上先返回的并不是
37. \* 整个程序运行的时间，也不可能做到，实质返回的是一个{elapsed\_time}标签，然后
38. \* Output在处理输出的时候，再计算出整个程序运行时间，因为处理输出阶段程序可以视
39. \* 为处于最后阶段，于是可以近似计算出总时间，然后把输出中的{elapsed\_time}替换掉。
40. \* 下面的memory\_usage()原理相同。
41. \* 详见：core/Output.php 中的\_display()方法。
42. \*/
43. if ($point1 == '')
44. {
45. return '{elapsed\_time}';
46. }
48. if ( ! isset($this->marker[$point1]))
49. {
50. return '';
51. }
53. if ( ! isset($this->marker[$point2]))
54. {
55. $this->marker[$point2] = microtime();
56. }
58. //这里为什么要用到list，是因为microtime();返回值是“msec sec”的格式。
59. list($sm, $ss) = explode(' ', $this->marker[$point1]);
60. list($em, $es) = explode(' ', $this->marker[$point2]);
62. return number\_format(($em + $es) - ($sm + $ss), $decimals);
63. }
65. // --------------------------------------------------------------------
67. /\*\*
68. \* Memory Usage
69. \*/
70. function memory\_usage()
71. {
72. //Output::\_display();
73. return '{memory\_usage}';
74. }
76. }

[CodeIgniter源码分析之Model.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Model.php%20–%20Calix的博客_files\CodeIgniter源码分析之Model.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. /\*\*
3. \* CodeIgniter Model Class
4. \*/
5. class CI\_Model {
7. /\*\*
8. \* Constructor
9. \*/
10. function \_\_construct()
11. {
12. log\_message('debug', "Model Class Initialized");
13. }
15. /\*\*
16. \* \_\_get
17. \* Model类的代码也非常少，有用的方法就下面这一个，下面这个方法是为了在Model里面可以像控制器那么通过$this->
18. \* 做很多事情。例如想在model里面加载某个library，就可以$this->load->library(xxx)，其实它都是盗用controller的。
19. \*/
20. function \_\_get($key)
21. {
22. $CI =& get\_instance();
23. return $CI->$key;
24. }
25. }

[CodeIgniter源码分析之Controller.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Controller.php%20–%20Calix的博客_files\CodeIgniter源码分析之Controller.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
3. /\*\*
4. \* CodeIgniter Application Controller Class
5. \*/
6. class CI\_Controller {
8. private static $instance;
10. /\*\*
11. \* Constructor
12. \*/
13. public function \_\_construct()
14. {
15. //通过self::$instance实现单例化，在第一次实例时，这个静态变量实质就是引用了这个实例。
16. //以后都可以通过&get\_instance();来获得这个单一实例。
17. self::$instance =& $this;
19. //把目前程序已经加载的所有的组件都给这个超级控制器来掌管。
20. foreach (is\_loaded() as $var => $class)
21. {
22. $this->$var =& load\_class($class);
23. }
25. //给超级控制器加载Loader组件，这个组件是它的好助手，很多时候你会经常用到$this->load->xxx()的形式加载某个东西，
26. //这个load就是控制器被构造的时候就伴随存在的。
27. $this->load =& load\_class('Loader', 'core');
29. //初始化Loader组件，详细Loader.php
30. $this->load->initialize();
32. log\_message('debug', "Controller Class Initialized");
33. }
35. public static function &get\_instance()
36. {
37. return self::$instance;
38. }
39. }

[CodeIgniter源码分析之Exception.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Exception.php%20–%20Calix的博客_files\CodeIgniter源码分析之Exception.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. // ------------------------------------------------------------------------
4. /\*\*
5. \* Exceptions Class
6. \*/
7. class CI\_Exceptions {
8. var $action;
9. var $severity;
10. var $message;
11. var $filename;
12. var $line;
14. /\*\*
15. \* Nesting level of the output buffering mechanism
16. \*/
17. var $ob\_level;
19. /\*\*
20. \* List if available error levels
21. \*/
22. var $levels = array(
23. E\_ERROR => 'Error',
24. E\_WARNING => 'Warning',
25. E\_PARSE => 'Parsing Error',
26. E\_NOTICE => 'Notice',
27. E\_CORE\_ERROR => 'Core Error',
28. E\_CORE\_WARNING => 'Core Warning',
29. E\_COMPILE\_ERROR => 'Compile Error',
30. E\_COMPILE\_WARNING => 'Compile Warning',
31. E\_USER\_ERROR => 'User Error',
32. E\_USER\_WARNING => 'User Warning',
33. E\_USER\_NOTICE => 'User Notice',
34. E\_STRICT => 'Runtime Notice'
35. );

38. /\*\*
39. \* Constructor
40. \*/
41. public function \_\_construct()
42. {
43. $this->ob\_level = ob\_get\_level();
44. // Note: Do not log messages from this constructor.
45. }
47. // --------------------------------------------------------------------
49. /\*\*
50. \* Exception Logger
51. \*/
52. function log\_exception($severity, $message, $filepath, $line)
53. {
54. $severity = ( ! isset($this->levels[$severity])) ? $severity : $this->levels[$severity];
56. log\_message('error', 'Severity: '.$severity.' --> '.$message. ' '.$filepath.' '.$line, TRUE);
57. }
59. // --------------------------------------------------------------------
61. /\*\*
62. \* 404 Page Not Found Handler
63. \*/
64. function show\_404($page = '', $log\_error = TRUE)
65. {
66. $heading = "404 Page Not Found";
67. $message = "The page you requested was not found.";
69. // By default we log this, but allow a dev to skip it
70. if ($log\_error)
71. {
72. log\_message('error', '404 Page Not Found --> '.$page);
73. }
75. echo $this->show\_error($heading, $message, 'error\_404', 404);
76. exit;
77. }
79. // --------------------------------------------------------------------
81. /\*\*
82. \* 先解释一下show\_php\_error,show\_error,和show\_404之间的关系和区别。
83. \* show\_php\_error()是代码本身的一些错误，例如变量未定义之类的，平时我们调试的时候经常见到的一些错误，是不小心写错代码而导致的。
84. \* show\_error()是有意识触发的错误，不是代码写错，而是代码不当，或者用户操作不当，比如找不到控制器，指定方法之类的，CI就show一
85. \* 个错误出来，当然开发者也可以调用此方法响应一个错误信息，某种程度上类似于catch到一个exception之后的处理，然后根据exception
86. \* 发出不同的提示信息。
87. \* show\_404()是show\_error()中的一种特殊情况，就是请求不存在的情况，响应一个404错误。
88. \*/
89. /\*\*
90. \* General Error Page
91. \*/
92. function show\_error($heading, $message, $template = 'error\_general', $status\_code = 500)
93. {
94. //默认是500，内部服务错误。是指由于程序代码写得不恰当而引起的，因此向浏览器回应一个内部错误。
95. set\_status\_header($status\_code);
97. $message = '<p>'.implode('</p><p>', ( ! is\_array($message)) ? array($message) : $message).'</p>';
99. //此处的作用与show\_php\_error()中的相应位置作用一样。
100. if (ob\_get\_level() > $this->ob\_level + 1)
101. {
102. ob\_end\_flush();
103. }
104. ob\_start();
105. include(APPPATH.'errors/'.$template.'.php');
106. $buffer = ob\_get\_contents();
107. ob\_end\_clean();
108. //这里是return，因为一般情况下，是使用core/Common.php中，全局函数show\_error()间接使用当前Exception::show\_error()方法。
109. return $buffer;
110. }
112. // --------------------------------------------------------------------
114. /\*\*
115. \* Native PHP error handler
116. \*/
117. function show\_php\_error($severity, $message, $filepath, $line)
118. {
119. //取得对应错误级别相对的说明。在$this->levels中定义。
120. $severity = ( ! isset($this->levels[$severity])) ? $severity : $this->levels[$severity];
122. $filepath = str\_replace("\\", "/", $filepath);
124. //为了安全起见，只显示错误文件最后两段路径信息。
125. if (FALSE !== strpos($filepath, '/'))
126. {
127. $x = explode('/', $filepath);
128. $filepath = $x[count($x)-2].'/'.end($x);
129. }
131. /\*
132. \* 如果还没看过core/Loader.php，下面这个判断可能让人有点迷惑。
133. \* ob\_get\_level()是取得当前缓冲机制的嵌套级别。（缓冲是可以一层嵌一层的。）
134. \* 右边的$this->ob\_level是在\_\_construct()里面同样通过ob\_get\_level()被赋值的。
135. \* 也就是说，有可能出现：Exception组件被加载时（也就是应用刚开始运行时）的缓冲级别（其实也就是程序最开始的时候的缓冲级别，那时
136. \* 候是还没有ob\_start()过的），与发生错误的时候的缓冲级别相差1。
137. \* 在控制器执行$this->load->view("xxx");的时候，实质，Loader引入并执行这个视图文件的时候，是先把缓冲打开，即
138. \* 先ob\_start()，所有输出放到缓冲区（详见：core/Loader.php中的\_ci\_load()）,然后再由Output处理输出。
139. \* 因此，如果是在视图文件发生错误，则就会出现缓冲级别相差1的情况，此时先把输出的内容给flush出来，然后再把错误信息输出。
140. \*
141. \*/
142. if (ob\_get\_level() > $this->ob\_level + 1)
143. {
144. ob\_end\_flush();
145. }
146. ob\_start();
147. //错误信息模板，位于应用目录errors/下。
148. include(APPPATH.'errors/error\_php.php');
149. $buffer = ob\_get\_contents();
150. ob\_end\_clean();
151. echo $buffer;
152. }
153. }

[CodeIgniter源码分析之Router.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Router.php%20–%20Calix的博客_files\CodeIgniter源码分析之Router.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
3. // ------------------------------------------------------------------------
5. /\*\*
6. \* Router Class
7. \*/
8. class CI\_Router {
10. /\*\*
11. \* Config class
12. \*/
13. var $config;
15. /\*\*
16. \* List of routes
18. \*/
19. var $routes = array();
21. /\*\*
22. \* List of error routes
23. \*/
24. var $error\_routes = array();
26. /\*\*
27. \* Current class name
28. \*/
29. var $class = '';
31. /\*\*
32. \* Current method name
33. \*/
34. var $method = 'index';
36. /\*\*
37. \* Sub-directory that contains the requested controller class
38. \*/
39. var $directory = '';
41. /\*\*
42. \* Default controller (and method if specific)
43. \*/
44. var $default\_controller;
46. /\*\*
47. \* Constructor
48. \*/
49. function \_\_construct()
50. {
51. $this->config =& load\_class('Config', 'core');
52. $this->uri =& load\_class('URI', 'core');
53. log\_message('debug', "Router Class Initialized");
54. }
56. // --------------------------------------------------------------------
58. /\*\*
59. \* Set the route mapping
60. \*/
61. function \_set\_routing()
62. {
64. //如果项目是允许通过query\_strings的形式，并且有通过$\_GET的方式请求控制器的话，则以query\_string形式，也就是
65. //?c=xx的形式确定路由。
66. $segments = array();
67. if ($this->config->item('enable\_query\_strings') === TRUE AND isset($\_GET[$this->config->item('controller\_trigger')]))
68. {
69. //上面这里为什么还要判断有没有通过get的方式指定控制器？其实是因为如果允许query\_string的形式请求路由，但是却
70. //没有通过query\_string（或者说是get）的形式指定路由的话（其实就说明这个通过query\_string方式的uri是无效的），
71. //此时,我们依然会采用“段”的形式。
73. //取得目录名,目录名，控制名和方法名传递的变量名都是可以自定义的，在config/config.php里面。
74. if (isset($\_GET[$this->config->item('directory\_trigger')]))
75. {
76. $this->set\_directory(trim($this->uri->\_filter\_uri($\_GET[$this->config->item('directory\_trigger')])));
77. $segments[] = $this->fetch\_directory();
78. }
80. //取得控制器名
81. if (isset($\_GET[$this->config->item('controller\_trigger')]))
82. {
83. $this->set\_class(trim($this->uri->\_filter\_uri($\_GET[$this->config->item('controller\_trigger')])));
84. $segments[] = $this->fetch\_class();
85. }
87. //取得方法名
88. if (isset($\_GET[$this->config->item('function\_trigger')]))
89. {
90. $this->set\_method(trim($this->uri->\_filter\_uri($\_GET[$this->config->item('function\_trigger')])));
91. $segments[] = $this->fetch\_method();
92. }
93. }
94. //。。。。。。。。。。。。。。。位置1
96. // Load the routes.php file.
97. //引入关于路由方面的配置信息。配置文件里面是一个名字为$route的数组。
98. if (defined('ENVIRONMENT') AND is\_file(APPPATH.'config/'.ENVIRONMENT.'/routes.php'))
99. {
100. include(APPPATH.'config/'.ENVIRONMENT.'/routes.php');
101. }
102. elseif (is\_file(APPPATH.'config/routes.php'))
103. {
104. include(APPPATH.'config/routes.php');
105. }
107. //下面这个莫名出现的$route变量(注意不是$this->routes哦)，就是写在配置文件里面。把它copy到$this->routes。
108. //这个$routes是用来指定默认控制器和默认方法，404（请求路由不存在）后规定的路由以及一些路由重定向（？这个重写向的
109. //实现在Router::\_parse\_routes()中实现）的信息。
110. $this->routes = ( ! isset($route) OR ! is\_array($route)) ? array() : $route;
111. unset($route);//利用完就干掉。
113. //根据刚才的配置信息，设定默认控制器，没有的话，就为FLASE。
114. $this->default\_controller = ( ! isset($this->routes['default\_controller']) OR $this->routes['default\_controller'] == '') ? FALSE : strtolower($this->routes['default\_controller']);
116. //这个判断的位置放得有点怪，我觉得可以放到上面“位置1”的地方，下面的代码是判断刚才有没有通过query\_string的方式拿到
117. //路由信息，如果拿到的话，那么就不再尝试“段”的方式确定路由了。直接确定路由，结束本函数。
118. if (count($segments) > 0)
119. {
120. //\_validate\_quest($segments);的作用就是确定并设置路由。
121. //这个函数执行完之后，Router::$class,Router::$directory（如果有）都会有相应值。
122. return $this->\_validate\_request($segments);
123. }
125. //下面的\_fetch\_uri\_string()详见：URI.php，在这个位置只需知道它的作用是：
126. //从uri中检测处理，把我们确定路由需要的信息（例如“index.php/index/welcome/1”后
127. //面"index/welcome/1"这串）放到$this->uri->uri\_string这个东西中。
128. $this->uri->\_fetch\_uri\_string();
130. //上面\_fetch\_uri\_string()完了之后，这个uri\_string就会有我们要用的信息，如果为空的话，那么就用把路由设置为默认的。
131. if ($this->uri->uri\_string == '')
132. {
133. //移步至Router::\_set\_default\_controller();
134. return $this->\_set\_default\_controller();
135. }
137. //如果$this->uri->uri\_string 不为空，那么，会通过下面的方式确定路由
139. //这里只是简单地把后缀去掉而已，因为CI允许在uri后面加后缀，但它其实对我们寻找路由是多余，而且会造成影响的，所以先去掉。
140. $this->uri->\_remove\_url\_suffix();
142. //把最初的uri，变成数组放在segments里面。
143. $this->uri->\_explode\_segments();
145. //开始找路由。移步至 Router::\_parse\_routes();
146. $this->\_parse\_routes();
148. //设置为由1开始。
149. $this->uri->\_reindex\_segments();
150. }
152. // --------------------------------------------------------------------
154. /\*\*
155. \* Set the default controller
156. \*/
157. function \_set\_default\_controller()
158. {
159. //在Router::\_set\_routing()函数里面有一个操作，是从配置文件里面读取默认控制器名，如果没有就有FALSE。
160. if ($this->default\_controller === FALSE)
161. {
162. //如果没有默认的话，就报错，结束程序。
163. //实质上，这个\_set\_default\_controller()仅仅是在uri没有指定控制器，要求访问默认控制器的时候才
164. //被调用，所以如果连默认控制器都没有，那么可以果断报错。
165. show\_error("Unable to determine what should be displayed. A default route has not been specified in the routing file.");
166. }
168. //如果有，下面我们就来把默认的控制器设置为当前要找的路由。
170. //这里只是分“有指定默认方法”和“没有指定”两种情况而已。不过要弄点下面那个$this->\_set\_request($x);CI这几个函数
171. //也许写得很妙，但是让人看得纠结。。。
172. if (strpos($this->default\_controller, '/') !== FALSE)
173. {
174. $x = explode('/', $this->default\_controller);
176. $this->set\_class($x[0]);
177. $this->set\_method($x[1]);
178. $this->\_set\_request($x);//移步至Router::\_set\_request();
179. }
180. else
181. {
182. $this->set\_class($this->default\_controller);
183. $this->set\_method('index');
184. $this->\_set\_request(array($this->default\_controller, 'index'));
185. }
187. // re-index the routed segments array so it starts with 1 rather than 0
188. $this->uri->\_reindex\_segments();
190. log\_message('debug', "No URI present. Default controller set.");
191. }
193. // --------------------------------------------------------------------
195. /\*\*
196. \* Set the Route
197. \*/
198. function \_set\_request($segments = array())
199. {
200. /\*\*
201. \* 下面来解剖一下这个让人纠结的函数。。第一次看的时候差点被它们这几个函数搞晕。
202. \*/
204. /\*\*
205. \* 看，这里有调用Router::\_validate\_request();而Router::\_validate\_request()的作用是检测寻找出一个
206. \* 正确存在的路由，并确定它，确定后的值分别放到Rouer::$class这些属性里面。所以使到这个\_set\_request()也有
207. \* 这种确定路由的功能。
208. \*
209. \* 注：
210. \* $segments=$this->\_validate\_request($segments); 等式右边，括号里面的这个$segments，也就是调用
211. \* \_set\_request()时传入来的这个参数，它有这样的特点：
212. \* 1）如果这时\_set\_request()是在Router::\_set\_default\_controller()中调用的话，那个这个$segments是永远不会为
213. \* 空数组，嗯，绝对不会。
214. \*
215. \*
216. \* 而左边这个$segments的值，经过下面这行代码后，要么为空数组array(),要么为确定路由后的段数组。
217. \* 为空数组的原因是，$this->\_validate\_request();里面没有找到当前目录的默认控制器。此时，右边的
218. \* $segments要么为空，要么只指定了目录但默认控制器不存在。
219. \*/
220. $segments = $this->\_validate\_request($segments);
222. if (count($segments) == 0)
223. {
224. //所以如果上面返回了空数组，就会进到这里。
225. //这里居然又调回了\_set\_default\_controller()! 坑爹吧！
226. return $this->\_set\_default\_controller();
227. /\*\*
228. \* 我曾经想过，下面这里会不会死循环：
229. \* 假如，我在配置文件里面的默认控制器设为welcome，然后controllers/下没有welcome.php,但controllers/下有
230. \* welcome/有这个目录（里面没东西），然后通过http://localhost/CI/来访问默认控制器，那会怎样呢？
231. \* 首先，它会进入\_set\_routing();然后发现$this->uri->uri\_string为空，进入\_set\_default\_controller();
232. \* 然后发现在\_set\_default\_controller里，发现$this->default\_controller不为FALSE，（＠＠＠＠），然后再
233. \* 进入这\_set\_request()里面，再进入\_validate\_request()里面，会不会\_validate\_request里返回空数组？因为
234. \* 指定了目录，没有指定控制器，访问默认的，又不存在，然后返回空数组，返回空数组后，最终就会走来你正在看的这个位置，
235. \* 然后这个位置再调用\_set\_default\_controller();然后死循环了。。。
236. \*
237. \* 答案是不会的。
238. \* 原因在于：
239. \* 我们回到上面解译那个（＠＠＠＠）的地方，在这里，发现$this->default\_controller不为FALSE后，它会进入这个else
240. \* 里面
241. \* else
242. \* {
243. \* $this->set\_class($this->default\_controller); ..............1
244. \* $this->set\_method('index'); ...................2
245. \* $this->\_set\_request(array($this->default\_controller, 'index')); ..........3
246. \* }
247. \*
248. \* 然后第3行，传入\_set\_request($segments)中的那个$segments其实是
249. \* array('welcome','index')，重点在于那个小小的'index'!!!!!!!
250. \* 这样一来，我们进入\_validate\_request()的时候，我们实质并没有“指定目录但没有指定控制器，访问默认控制器”，
251. \* 而是“指定了一个welcome的目录，和一个叫index的控制器！！”，所以才不会死循环。
252. \* 如果你试着把第3行那个'index'去掉，那么，一定会死循环！！！！！！！！不信试试！CI太牛逼了，居然这样做。汗。。
253. \* 当然，‘index’还有一个作用，就是设置默认方法啦。
254. \*/
255. }
257. $this->set\_class($segments[0]);
259. if (isset($segments[1]))
260. {
261. // A standard method request
262. $this->set\_method($segments[1]);
263. }
264. else
265. {
266. $segments[1] = 'index';
267. }
269. //这里要说一下，现在是在ROUTER里面为URI赋值，URI里面的这个URI::$rsegments是经过处理，并确定路由后，实质调用的路由的段信息。
270. //而URI::$segments （前面少了个r），则是原来没处理前的那个，即直接由网址上面得出来的那个。
271. $this->uri->rsegments = $segments;
272. }
274. // --------------------------------------------------------------------
276. /\*\*
277. \* Validates the supplied segments. Attempts to determine the path to
278. \* the controller.
279. \*/
280. function \_validate\_request($segments)
281. {
282. if (count($segments) == 0)
283. {
284. return $segments;
285. }
287. if (file\_exists(APPPATH.'controllers/'.$segments[0].'.php'))
288. {
289. //如果直接在controllers这个目录下找到与第一段相应的控制器名，那就说明找到了控制器，确定路由，返回。
290. return $segments;
291. }
293. //如果上面没有找到，再看看这个“第一段”是不是一个目录，因为CI是允许控制器放在自定义的目录下的。
294. if (is\_dir(APPPATH.'controllers/'.$segments[0]))
295. {
296. // Set the directory and remove it from the segment array
297. //如果的确是目录，那么就可以确定路由的目录部分了。
298. $this->set\_directory($segments[0]);
299. //去掉目录部分。进一步进行路由寻找。
300. $segments = array\_slice($segments, 1);
302. //如果uri请求中除了目录还有其它“段”，那说明是有请求某指定控制器的。
303. if (count($segments) > 0)
304. {
305. //指定请求的控制器找不到的话，那只好报错了。
306. if ( ! file\_exists(APPPATH.'controllers/'.$this->fetch\_directory().$segments[0].'.php'))
307. {
308. //报错也有两方式，一种是默认的，一种是自义定的。
309. //下面这个404\_override就是在config/routes.php定义的一个路由找不到时候的默认处理控制器了，如果有定义
310. //我们调用它。
311. if ( ! empty($this->routes['404\_override']))
312. {
313. $x = explode('/', $this->routes['404\_override']);
315. $this->set\_directory('');//把刚才设置好的路由的目录部分去掉，因为现在路由是我们定义的404路由。
316. $this->set\_class($x[0]);//这里可以看出，我们定义的404路由是不允许放在某个目录下的，只能直接放在controllers/下
317. $this->set\_method(isset($x[1]) ? $x[1] : 'index');//默认是index方法
319. return $x; //同样，返回“段”数组
320. }
321. else
322. {
323. //默认找不到路由的方法。在core/Common.php中定义的全局函数（实质调用Exception组件进行处理）。
324. show\_404($this->fetch\_directory().$segments[0]);
325. }
326. }
327. }
328. else
329. {
330. //来到这里，说明了是uri请求指定了目录，而没有指定控制器的情况下。那么，我们默认当前路由是在当前目录下请求默认的
331. //控制器和方法。
333. //下面这个判断只是判断一下$this->default\_controller有没有指定方法而已。
334. if (strpos($this->default\_controller, '/') !== FALSE)
335. {
336. $x = explode('/', $this->default\_controller);
337. $this->set\_class($x[0]);
338. $this->set\_method($x[1]);
339. }
340. else
341. {
342. $this->set\_class($this->default\_controller);
343. $this->set\_method('index');//没有的话就默认为index方法。
344. }
346. // Does the default controller exist in the sub-folder?
347. //如果连默认控制器都不存在的话，就无语了，说明uri打算请求这个目录的默认控制器，结果没有这个默认控制器，那暂时
348. //返回个空数组。（但是看清楚，上面已经$this->set\_class()了，说明即使没有，我们也已经把默认控制器的名字算
349. //是确定下来先）
350. if ( ! file\_exists(APPPATH.'controllers/'.$this->fetch\_directory().$this->default\_controller.'.php'))
351. {
352. $this->directory = '';
353. return array();
354. }
356. }
358. //能够有命来到这一步，是说明的确是某个目录下找到了控制器，或者是找到了定义的默认控制器。
359. //但是注意，这个$segments返回的“段”信息都是不包括目录的。它是一个数组形式，第一个元素是控制器名。
360. //例如：array('acontroller','amethod','xx','xx')。。
361. return $segments;
362. }
364. //来到这里，就说明了，即找不到controllers/下相应的控制器，也找不到这样的目录。那就报错咯。
365. if ( ! empty($this->routes['404\_override']))
366. {
367. $x = explode('/', $this->routes['404\_override']);
369. $this->set\_class($x[0]);
370. $this->set\_method(isset($x[1]) ? $x[1] : 'index');
372. return $x;
373. }
375. // Nothing else to do at this point but show a 404
376. show\_404($segments[0]);
377. }
379. // --------------------------------------------------------------------
381. /\*\*
382. \* Parse Routes
383. \*/
384. function \_parse\_routes()
385. {
386. // Turn the segment array into a URI string
387. //知道\_set\_request()是干嘛的之后，下面的条理就比较清晰了。
388. $uri = implode('/', $this->uri->segments);
390. // Is there a literal match? If so we're done
391. if (isset($this->routes[$uri]))
392. {
393. return $this->\_set\_request(explode('/', $this->routes[$uri]));
394. }
396. /\*\*
397. \* CI有路由重定向的功能，重定向的规则和实现就是在这里。
398. \*/
399. foreach ($this->routes as $key => $val)
400. {
401. $key = str\_replace(':any', '.+', str\_replace(':num', '[0-9]+', $key));
403. if (preg\_match('#^'.$key.'$#', $uri))
404. {
405. // Do we have a back-reference?
406. if (strpos($val, '$') !== FALSE AND strpos($key, '(') !== FALSE)
407. {
408. $val = preg\_replace('#^'.$key.'$#', $val, $uri);
409. }
411. return $this->\_set\_request(explode('/', $val));
412. }
413. }
414. $this->\_set\_request($this->uri->segments);
415. }
417. // --------------------------------------------------------------------
419. /\*\*
420. \* Set the class name
421. \*/
422. function set\_class($class)
423. {
424. $this->class = str\_replace(array('/', '.'), '', $class);
425. }
427. // --------------------------------------------------------------------
429. /\*\*
430. \* Fetch the current class
431. \*/
432. function fetch\_class()
433. {
434. return $this->class;
435. }
437. // --------------------------------------------------------------------
439. /\*\*
440. \* Set the method name
441. \*/
442. function set\_method($method)
443. {
444. $this->method = $method;
445. }
447. // --------------------------------------------------------------------
449. /\*\*
450. \* Fetch the current method
451. \*/
452. function fetch\_method()
453. {
454. if ($this->method == $this->fetch\_class())
455. {
456. return 'index';
457. }
459. return $this->method;
460. }
462. // --------------------------------------------------------------------
464. /\*\*
465. \* Set the directory name
466. \*/
467. function set\_directory($dir)
468. {
469. $this->directory = str\_replace(array('/', '.'), '', $dir).'/';
470. }
472. // --------------------------------------------------------------------
474. /\*\*
475. \* Fetch the sub-directory (if any) that contains the requested controller class
476. \*/
477. function fetch\_directory()
478. {
479. return $this->directory;
480. }
482. // --------------------------------------------------------------------
484. /\*\*
485. \* Set the controller overrides
486. \*/
487. function \_set\_overrides($routing)
488. {
489. if ( ! is\_array($routing))
490. {
491. return;
492. }
494. if (isset($routing['directory']))
495. {
496. $this->set\_directory($routing['directory']);
497. }
499. if (isset($routing['controller']) AND $routing['controller'] != '')
500. {
501. $this->set\_class($routing['controller']);
502. }
504. if (isset($routing['function']))
505. {
506. $routing['function'] = ($routing['function'] == '') ? 'index' : $routing['function'];
507. $this->set\_method($routing['function']);
508. }
509. }
511. }

[CodeIgniter源码分析之URI.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之URI.php%20–%20Calix的博客_files\CodeIgniter源码分析之URI.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. // ------------------------------------------------------------------------
4. /\*\*
5. \* URI Class
6. \*/
7. class CI\_URI {
9. /\*\*
10. \* List of cached uri segments
11. \*/
12. var $keyval = array();

15. /\*\*
16. \* Current uri string
17. \*/
18. var $uri\_string;

21. /\*\*
22. \* List of uri segments
23. \*/
24. var $segments = array();

27. /\*\*
28. \* Re-indexed list of uri segments
29. \*/
30. var $rsegments = array();
32. /\*\*
33. \* Constructor
34. \*/
35. function \_\_construct()
36. {
37. $this->config =& load\_class('Config', 'core');
38. log\_message('debug', "URI Class Initialized");
39. }

42. // --------------------------------------------------------------------
44. /\*\*
45. \* Get the URI String
46. \*/
47. //URI组件里面有很多方法，大都是一些辅助作用的方法，而此方法是URI最主线的一个方法。
48. function \_fetch\_uri\_string()
49. {
50. //下面的uri\_protocol是在config.php里面的一个配置项，其实是问你用哪种方式去检测uri的信息的意思，
51. //默认是AUTO，自动检测，也就是通过各种方式检测，直至检测到，或者全部方式都检测完。。
52. if (strtoupper($this->config->item('uri\_protocol')) == 'AUTO')
53. {
54. //开始尝试各种方式,主要有：命令行，REQUEST\_URI, PATH\_INFO, QUERY\_STRING.
56. //下面会多次出现$this->\_set\_uri\_string($str)这个方法，这个方法没别的，就是把$str经过
57. //过滤和修剪后值给$this->uri\_string属性，在这里暂时可以理解为就是赋值。
59. //如果脚本是在命令行模式下运行的话，那么参数就是通过$\_SERVER['argv']来传递。下面的
60. //$this->\_parse\_cli\_args();就是拿到符合我们需要的路由相关的一些参数鸟～如果大部分
61. //情况你没用命令行执行脚本的话，下面这个if暂时可以不用管。
62. if (php\_sapi\_name() == 'cli' or defined('STDIN'))
63. {
64. $this->\_set\_uri\_string($this->\_parse\_cli\_args());
65. return;
66. }
68. //这种REQUEST\_URI方式相对复杂一点，因此封装在$this->\_detect\_uri()；里面。
69. //其实大多数情况下，利用REQUEST URI和SCRIPT NAME都会得到我们想要的路径信息了。
70. if ($uri = $this->\_detect\_uri())
71. {
72. $this->\_set\_uri\_string($uri);
73. return;
74. }
76. //PATH\_INFO方式，个人觉得这种方式最经济，只是不是每次请求都有$\_SERVER['PATH\_INFO']这个变量。
77. $path = (isset($\_SERVER['PATH\_INFO'])) ? $\_SERVER['PATH\_INFO'] : @getenv('PATH\_INFO');
78. if (trim($path, '/') != '' && $path != "/".SELF)
79. {
80. $this->\_set\_uri\_string($path);
81. return;
82. }
84. //如果是用QUERY\_STRING的话，路径格式一般为index.php?/controller/method/xxx/xxx
85. $path = (isset($\_SERVER['QUERY\_STRING'])) ? $\_SERVER['QUERY\_STRING'] : @getenv('QUERY\_STRING');
86. if (trim($path, '/') != '')
87. {
88. $this->\_set\_uri\_string($path);
89. return;
90. }
92. //上面的方法都不行，那真是奇怪了。。所以尝试最后一种奇葩的方法，就是从$\_GET里面把那个键名拿出来。
93. if (is\_array($\_GET) && count($\_GET) == 1 && trim(key($\_GET), '/') != '')
94. {
95. $this->\_set\_uri\_string(key($\_GET));
96. return;
97. }
99. // We've exhausted all our options...
100. $this->uri\_string = '';
101. return;
102. }
104. //厄，这里是因为上面那个获得uri\_protocol配置的语句写在if里面，然后又没赋值到某个变量，所以这里要再写一次了
105. //可能是因为大多数情况下，我们都是选择AUTO方式吧。但是，这样写又何必呢。。
106. $uri = strtoupper($this->config->item('uri\_protocol'));
108. //其实就是按规定的方式去找路径而已。。
109. if ($uri == 'REQUEST\_URI')
110. {
111. $this->\_set\_uri\_string($this->\_detect\_uri());
112. return;
113. }
114. elseif ($uri == 'CLI')
115. {
116. $this->\_set\_uri\_string($this->\_parse\_cli\_args());
117. return;
118. }
120. //如果你在配置文件config.php里面把这个uri\_protocol定义成一种上面都没有的方式，那么就会执行下面的代码。
121. //意思是，就看$\_SERVER有没有这个uri\_protocol的变量了，有就给，没有就拉倒。
122. $path = (isset($\_SERVER[$uri])) ? $\_SERVER[$uri] : @getenv($uri);
123. $this->\_set\_uri\_string($path);
124. }
126. // --------------------------------------------------------------------
128. /\*\*
129. \* Set the URI String
130. \*/
131. function \_set\_uri\_string($str)
132. {
133. // Filter out control characters
134. $str = remove\_invisible\_characters($str, FALSE);
136. // If the URI contains only a slash we'll kill it
137. $this->uri\_string = ($str == '/') ? '' : $str;
138. }
140. // --------------------------------------------------------------------
142. /\*\*
143. \* Detects the URI
144. \*/
145. private function \_detect\_uri()
146. {
147. //如果这两个值缺少其中一个，那么这种方法行不通。
148. if ( ! isset($\_SERVER['REQUEST\_URI']) OR ! isset($\_SERVER['SCRIPT\_NAME']))
149. {
150. return '';
151. }
153. $uri = $\_SERVER['REQUEST\_URI'];//取得request\_uri
155. //注意下面这个是===0不是false! 接下来这个if 和下面的elseif分别是script\_name有文件名和没有文件名（如
156. //http://abc.com/CI/或者http://abc.com/CI/?c=index&m=welcome等）的不同情况的处理。
157. if (strpos($uri, $\_SERVER['SCRIPT\_NAME']) === 0)
158. {
159. //去掉共同部分，取得对路由有用的部分。
160. $uri = substr($uri, strlen($\_SERVER['SCRIPT\_NAME']));
161. }
162. elseif (strpos($uri, dirname($\_SERVER['SCRIPT\_NAME'])) === 0)
163. {
164. //作用同上
165. $uri = substr($uri, strlen(dirname($\_SERVER['SCRIPT\_NAME'])));
166. }
168. //这里是为兼容?/abc/xx/的形式。
169. if (strncmp($uri, '?/', 2) === 0)
170. {
171. $uri = substr($uri, 2);
172. }
174. //在这里$uri可能是?xxx＝xx的形式，也可能是直接xxx=xx，也可能是/
175. $parts = preg\_split('#\?#i', $uri, 2);
176. $uri = $parts[0];
177. //如果是能通过上述的正则分割出两段，那么，是通过query\_string即？的形式进行路由访问
178. if (isset($parts[1]))
179. {
180. $\_SERVER['QUERY\_STRING'] = $parts[1];
181. parse\_str($\_SERVER['QUERY\_STRING'], $\_GET);
182. }
183. else
184. {
185. $\_SERVER['QUERY\_STRING'] = '';
186. $\_GET = array();
187. }
189. //如果为/，或者为空，有两种情况，要么就是通过query\_string，所以此时$parts[0]就是等于下面两种可能，同时我们
190. //已经通过$parts[1]拿到要拿的信息，则可以返回。要么就是以段的形式，但是段的信息为空，即直接访问入口文件而没有
191. //任何路由信息的传递，也可以直接返回。
192. if ($uri == '/' || empty($uri))
193. {
194. return '/';
195. }
197. //这里我个人觉得是上面的strpos($uri, $\_SERVER['SCRIPT\_NAME']) === 0和elseif都无法匹配的时候，
198. //返回这个url的path部分。
199. $uri = parse\_url($uri, PHP\_URL\_PATH);
201. // Do some final cleaning of the URI and return it
202. return str\_replace(array('//', '../'), '/', trim($uri, '/'));
203. }
205. // --------------------------------------------------------------------
207. /\*\*
208. \* Parse cli arguments
209. \*/
210. private function \_parse\_cli\_args()
211. {
212. //返回在命令行模式下运行时传递的参数。
213. $args = array\_slice($\_SERVER['argv'], 1);//因为第一个参数是当前文件名，所以从第二个开始才是我们要获取的。
215. //返回一个由'/'字符串拼接的字符串，因为$this->uri\_string是一个字符串。
216. return $args ? '/' . implode('/', $args) : '';
217. }
219. // --------------------------------------------------------------------
221. /\*\*
222. \* Filter segments for malicious characters
223. \*/
224. //过滤不合法字符
225. function \_filter\_uri($str)
226. {
227. if ($str != '' && $this->config->item('permitted\_uri\_chars') != '' && $this->config->item('enable\_query\_strings') == FALSE)
228. {
229. // preg\_quote() in PHP 5.3 escapes -, so the str\_replace() and addition of - to preg\_quote() is to maintain backwards
230. // compatibility as many are unaware of how characters in the permitted\_uri\_chars will be parsed as a regex pattern
231. if ( ! preg\_match("|^[".str\_replace(array('\\-', '\-'), '-', preg\_quote($this->config->item('permitted\_uri\_chars'), '-'))."]+$|i", $str))
232. {
233. show\_error('The URI you submitted has disallowed characters.', 400);
234. }
235. }
237. // Convert programatic characters to entities
238. $bad = array('$', '(', ')', '%28', '%29');
239. $good = array('&#36;', '&#40;', '&#41;', '&#40;', '&#41;');
241. return str\_replace($bad, $good, $str);
242. }
244. // --------------------------------------------------------------------
246. /\*\*
247. \* Remove the suffix from the URL if needed
248. \*/
249. //去掉url的我们自定义的后缀。
250. function \_remove\_url\_suffix()
251. {
253. if ($this->config->item('url\_suffix') != "")
254. {
255. $this->uri\_string = preg\_replace("|".preg\_quote($this->config->item('url\_suffix'))."$|", "", $this->uri\_string);
256. }
257. }
259. // --------------------------------------------------------------------
261. /\*\*
262. \* Explode the URI Segments. The individual segments will
263. \* be stored in the $this->segments array.
264. \*/
265. //把uri\_string拆成段（同时对各段进行过滤），保存到URI::$segments中。
266. function \_explode\_segments()
267. {
268. foreach (explode("/", preg\_replace("|/\*(.+?)/\*$|", "\\1", $this->uri\_string)) as $val)
269. {
270. // Filter segments for security
271. $val = trim($this->\_filter\_uri($val));
273. if ($val != '')
274. {
275. $this->segments[] = $val;
276. }
277. }
278. }
280. // --------------------------------------------------------------------
281. /\*\*
282. \* Re-index Segments
283. \*/
284. //使得出来的段以下标1开始保存。
285. function \_reindex\_segments()
286. {
287. array\_unshift($this->segments, NULL);
288. array\_unshift($this->rsegments, NULL);
289. unset($this->segments[0]);
290. unset($this->rsegments[0]);
291. }
293. // --------------------------------------------------------------------
295. /\*\*
296. \* Fetch a URI Segment
297. \*/
298. //返回某一段
299. function segment($n, $no\_result = FALSE)
300. {
301. return ( ! isset($this->segments[$n])) ? $no\_result : $this->segments[$n];
302. }
304. // --------------------------------------------------------------------
306. /\*\*
307. \* Fetch a URI "routed" Segment
308. \*/
309. //返回确定路由后的某一段
310. function rsegment($n, $no\_result = FALSE)
311. {
312. return ( ! isset($this->rsegments[$n])) ? $no\_result : $this->rsegments[$n];
313. }
315. // --------------------------------------------------------------------
317. /\*\*
318. \* Generate a key value pair from the URI string
319. \*/
320. function uri\_to\_assoc($n = 3, $default = array())
321. {
322. return $this->\_uri\_to\_assoc($n, $default, 'segment');
323. }
324. /\*\*
325. \* Identical to above only it uses the re-routed segment array
326. \*/
327. function ruri\_to\_assoc($n = 3, $default = array())
328. {
329. return $this->\_uri\_to\_assoc($n, $default, 'rsegment');
330. }
332. // --------------------------------------------------------------------
334. /\*\*
335. \* Generate a key value pair from the URI string or Re-routed URI string
336. \*/
337. function \_uri\_to\_assoc($n = 3, $default = array(), $which = 'segment')
338. {
339. if ($which == 'segment')
340. {
341. $total\_segments = 'total\_segments';
342. $segment\_array = 'segment\_array';
343. }
344. else
345. {
346. $total\_segments = 'total\_rsegments';
347. $segment\_array = 'rsegment\_array';
348. }
350. if ( ! is\_numeric($n))
351. {
352. return $default;
353. }
355. if (isset($this->keyval[$n]))
356. {
357. return $this->keyval[$n];
358. }
360. if ($this->$total\_segments() < $n)
361. {
362. if (count($default) == 0)
363. {
364. return array();
365. }
367. $retval = array();
368. foreach ($default as $val)
369. {
370. $retval[$val] = FALSE;
371. }
372. return $retval;
373. }
375. $segments = array\_slice($this->$segment\_array(), ($n - 1));
377. $i = 0;
378. $lastval = '';
379. $retval = array();
380. foreach ($segments as $seg)
381. {
382. if ($i % 2)
383. {
384. $retval[$lastval] = $seg;
385. }
386. else
387. {
388. $retval[$seg] = FALSE;
389. $lastval = $seg;
390. }
392. $i++;
393. }
395. if (count($default) > 0)
396. {
397. foreach ($default as $val)
398. {
399. if ( ! array\_key\_exists($val, $retval))
400. {
401. $retval[$val] = FALSE;
402. }
403. }
404. }
406. // Cache the array for reuse
407. $this->keyval[$n] = $retval;
408. return $retval;
409. }
411. // --------------------------------------------------------------------
413. /\*\*
414. \* Generate a URI string from an associative array
415. \*/
416. function assoc\_to\_uri($array)
417. {
418. $temp = array();
419. foreach ((array)$array as $key => $val)
420. {
421. $temp[] = $key;
422. $temp[] = $val;
423. }
425. return implode('/', $temp);
426. }
428. // --------------------------------------------------------------------
430. /\*\*
431. \* Fetch a URI Segment and add a trailing slash
432. \*/
433. function slash\_segment($n, $where = 'trailing')
434. {
435. return $this->\_slash\_segment($n, $where, 'segment');
436. }
438. // --------------------------------------------------------------------
440. /\*\*
441. \* Fetch a URI Segment and add a trailing slash
442. \*/
443. function slash\_rsegment($n, $where = 'trailing')
444. {
445. return $this->\_slash\_segment($n, $where, 'rsegment');
446. }
448. // --------------------------------------------------------------------
450. /\*\*
451. \* Fetch a URI Segment and add a trailing slash - helper function
452. \*/
453. function \_slash\_segment($n, $where = 'trailing', $which = 'segment')
454. {
455. $leading = '/';
456. $trailing = '/';
458. if ($where == 'trailing')
459. {
460. $leading = '';
461. }
462. elseif ($where == 'leading')
463. {
464. $trailing = '';
465. }
467. return $leading.$this->$which($n).$trailing;
468. }
470. // --------------------------------------------------------------------
472. /\*\*
473. \* Segment Array
474. \*/
475. function segment\_array()
476. {
477. return $this->segments;
478. }
480. // --------------------------------------------------------------------
482. /\*\*
483. \* Routed Segment Array
484. \*/
485. function rsegment\_array()
486. {
487. return $this->rsegments;
488. }
490. // --------------------------------------------------------------------
492. /\*\*
493. \* Total number of segments
494. \*/
495. function total\_segments()
496. {
497. return count($this->segments);
498. }
500. // --------------------------------------------------------------------
502. /\*\*
503. \* Total number of routed segments
504. \*/
505. function total\_rsegments()
506. {
507. return count($this->rsegments);
508. }
510. // --------------------------------------------------------------------
512. /\*\*
513. \* Fetch the entire URI string
514. \*/
515. function uri\_string()
516. {
517. return $this->uri\_string;
518. }

521. // --------------------------------------------------------------------
523. /\*\*
524. \* Fetch the entire Re-routed URI string
525. \*/
526. function ruri\_string()
527. {
528. return '/'.implode('/', $this->rsegment\_array());
529. }
531. }

[CodeIgniter源码分析之Common.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之Common.php%20–%20Calix的博客_files\CodeIgniter源码分析之Common.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
3. // ------------------------------------------------------------------------
5. /\*\*
6. \* Common Functions
7. \*/
9. /\*\*
10. \* 为什么还要定义这些全局函数呢？比如说，下面有很多函数，如get\_config()、config\_item()这两个方法不是应该由
11. \* core/Config.php这个组件去做么？那个load\_class()不应该由core/Loader.php去做么？ 把这些函数定义出来貌似
12. \* 感觉架构变得不那么优雅，有点多余。
13. \* 其实是出于这样一种情况：
14. \* 比如说，如果一切和配置有关的动作都由Config组件来完成，一切加载的动作都由Loader来完成，
15. \* 试想一下，如果我要加载Config组件，那么，必须得通过Loader来加载，所以Loader必须比Config要更早实例化，
16. \* 但是如果Loader实例化的时候需要一些和Loader有关的配置信息才能实例化呢？那就必须通过Config来为它取得配置信息。
17. \* 这里就出现了鸡和鸡蛋的问题。。
18. \* 我之前写自己的框架也纠结过这样的问题，后来参考了YII框架，发现它里面其实都有同样的问题，它里面有个Exception的组件，
19. \* 但是在加载这个Exception组件之前，在加载其它组件的时候，如果出错了，那谁来处理异常和错误信息呢？答案就是先定义一些公共的函数。
20. \* 所以这些公共函数就很好地解决了这个问题，这也是为什么Common.php要很早被引入。
21. \*
22. \*/
24. // ------------------------------------------------------------------------
26. /\*\*
27. \* Determines if the current version of PHP is greater then the supplied value
28. \*/
29. if ( ! function\_exists('is\_php'))
30. {
31. //判断当前php版本是不是$version以上的。调用version\_compare()这个函数。
32. function is\_php($version = '5.0.0')
33. {
34. static $\_is\_php;
35. $version = (string)$version;
37. if ( ! isset($\_is\_php[$version]))
38. {
39. //PHP\_VERION能够获得当前php版本。
40. $\_is\_php[$version] = (version\_compare(PHP\_VERSION, $version) < 0) ? FALSE : TRUE;
41. }
43. return $\_is\_php[$version];
44. }
45. }
47. // ------------------------------------------------------------------------
49. /\*\*
50. \* Tests for file writability
51. \*/
52. if ( ! function\_exists('is\_really\_writable'))
53. {
54. //该函数和php官方手册上面写的差不多，兼容linux/Unix和windows系统：
55. //http://www.php.net/manual/en/function.is-writable.php
56. function is\_really\_writable($file)
57. {
59. //DIRECTORY\_SEPARATOR是系统的目录分割符。利用它的值可以知道当前是不是linux系统。
60. if (DIRECTORY\_SEPARATOR == '/' AND @ini\_get("safe\_mode") == FALSE)
61. {
62. //如果是linux系统的话，那么可以直接调用此方法来判断文件是否可写。
63. return is\_writable($file);
64. }
66. //如果是windows系统，则尝试写入一个文件来判断。
68. if (is\_dir($file))
69. {
70. //如果是目录，则创建一个随机命名的文件。
71. $file = rtrim($file, '/').'/'.md5(mt\_rand(1,100).mt\_rand(1,100));
73. //如果文件不能创建，则返回不可写。
74. if (($fp = @fopen($file, FOPEN\_WRITE\_CREATE)) === FALSE)
75. {
76. return FALSE;
77. }
79. fclose($fp);
80. //删除刚才的文件。
81. @chmod($file, DIR\_WRITE\_MODE);
82. @unlink($file);
83. return TRUE;
84. }
85. elseif ( ! is\_file($file) OR ($fp = @fopen($file, FOPEN\_WRITE\_CREATE)) === FALSE)
86. {
87. //如果是一个文件，而通过写入方式打不开，则返回不可写。
88. return FALSE;
89. }
91. fclose($fp);
92. return TRUE;
93. }
94. }
96. // ------------------------------------------------------------------------
98. /\*\*
99. \* Class registry
100. \*/
101. if ( ! function\_exists('load\_class'))
102. {
103. //加载类。默认是加载libraries里面的，如果要加载核心组件，$directory就为'core'
104. function &load\_class($class, $directory = 'libraries', $prefix = 'CI\_')
105. {
106. static $\_classes = array();//用一个静态数组，保存已经加载过的类的实例，防止多次实例消耗资源，实现单例化。
108. // Does the class exist? If so, we're done...
109. if (isset($\_classes[$class]))
110. {
111. return $\_classes[$class];//如果已经保存在这里，就返回它。
112. }
114. $name = FALSE;
116. //这里，如果应用目录下有和系统目录下相同的类的话，优先引入应用目录，也就是你自己定义的。
117. foreach (array(APPPATH, BASEPATH) as $path)
118. {
119. if (file\_exists($path.$directory.'/'.$class.'.php'))
120. {
121. $name = $prefix.$class;
123. if (class\_exists($name) === FALSE)
124. {
125. require($path.$directory.'/'.$class.'.php');
126. }
128. break;
129. }
130. }
132. //这里就用到的前缀扩展，如果在应用目录相应的目录下，有自己写的一些对CI库的扩展，那么我们加载的是它，而不是
133. //原来的。因为我们写的扩展是继承了CI原来的。
134. //所以可以看出，即使是CI的核心组件（core/下面的）我们都可以为之进行扩展。
135. if (file\_exists(APPPATH.$directory.'/'.config\_item('subclass\_prefix').$class.'.php'))
136. {
137. $name = config\_item('subclass\_prefix').$class;
139. if (class\_exists($name) === FALSE)
140. {
141. require(APPPATH.$directory.'/'.config\_item('subclass\_prefix').$class.'.php');
142. }
143. }
145. // Did we find the class?
146. if ($name === FALSE)
147. {
148. //这里用的是exit();来提示错误，而不是用show\_error();这是因为这个load\_class的错误有可能
149. //在加载Exception组件之前发。
150. exit('Unable to locate the specified class: '.$class.'.php');
151. }
153. // Keep track of what we just loaded
154. //这个函数只是用来记录已经被加载过的类的类名而已。
155. is\_loaded($class);
157. $\_classes[$class] = new $name();
158. return $\_classes[$class];
159. }
160. }
162. // --------------------------------------------------------------------
164. /\*\*
165. \* Keeps track of which libraries have been loaded. This function is
166. \* called by the load\_class() function above
167. \*/
168. if ( ! function\_exists('is\_loaded'))
169. {
170. //记录有哪些类是已经被加载的。
171. function is\_loaded($class = '')
172. {
173. static $\_is\_loaded = array();
175. if ($class != '')
176. {
177. $\_is\_loaded[strtolower($class)] = $class;
178. }
180. return $\_is\_loaded;
181. }
182. }
184. // ------------------------------------------------------------------------
186. /\*\*
187. \* Loads the main config.php file
188. \*/
189. if ( ! function\_exists('get\_config'))
190. {
191. //这个是读取配置信息的函数，在Config类被实例化之前，由它暂负责。
192. //而在Config类被实例化之前，我们需要读取的配置信息，其实仅仅是config.php这个主配置文件的。所以这个方法是不能读出
193. //config/下其它配置文件的信息的。
194. //这个$replace参数，是提供一个临时替换配置信息的机会，仅一次，因为执行一次后，配置信息都会保存在静态变量$\_config中，不能
195. //改变。
196. function &get\_config($replace = array())
197. {
198. static $\_config;
200. if (isset($\_config))
201. {
202. return $\_config[0];
203. }
205. // Is the config file in the environment folder?
206. if ( ! defined('ENVIRONMENT') OR ! file\_exists($file\_path = APPPATH.'config/'.ENVIRONMENT.'/config.php'))
207. {
208. $file\_path = APPPATH.'config/config.php';
209. }
211. // Fetch the config file
212. if ( ! file\_exists($file\_path))
213. {
214. exit('The configuration file does not exist.');
215. }
217. require($file\_path);
219. // Does the $config array exist in the file?
220. if ( ! isset($config) OR ! is\_array($config))
221. {
222. exit('Your config file does not appear to be formatted correctly.');
223. }
225. // Are any values being dynamically replaced?
226. if (count($replace) > 0)
227. {
228. foreach ($replace as $key => $val)
229. {
230. if (isset($config[$key]))
231. {
232. $config[$key] = $val;
233. }
234. }
235. }
237. return $\_config[0] =& $config;
238. }
239. }
241. // ------------------------------------------------------------------------
243. /\*\*
244. \* Returns the specified config item
245. \*/
246. if ( ! function\_exists('config\_item'))
247. {
248. //取得配置数组中某个元素。
249. function config\_item($item)
250. {
251. static $\_config\_item = array();
253. if ( ! isset($\_config\_item[$item]))
254. {
255. $config =& get\_config();
257. if ( ! isset($config[$item]))
258. {
259. return FALSE;
260. }
261. $\_config\_item[$item] = $config[$item];
262. }
264. return $\_config\_item[$item];
265. }
266. }
268. // ------------------------------------------------------------------------
270. /\*\*
271. \* Error Handler
272. \*/
274. //这里的show\_error和下面的show\_404以及 \_exception\_handler这三个错误的处理，实质都是由Exception组件完成的。
275. //详见core/Exception.php.
276. if ( ! function\_exists('show\_error'))
277. {
279. function show\_error($message, $status\_code = 500, $heading = 'An Error Was Encountered')
280. {
281. $\_error =& load\_class('Exceptions', 'core');
282. echo $\_error->show\_error($heading, $message, 'error\_general', $status\_code);
283. exit;
284. }
285. }
287. // ------------------------------------------------------------------------
289. /\*\*
290. \* 404 Page Handler
291. \*/
292. if ( ! function\_exists('show\_404'))
293. {
295. function show\_404($page = '', $log\_error = TRUE)
296. {
297. $\_error =& load\_class('Exceptions', 'core');
298. $\_error->show\_404($page, $log\_error);
299. exit;
300. }
301. }
303. // ------------------------------------------------------------------------
305. /\*\*
306. \* Error Logging Interface
307. \*/
308. if ( ! function\_exists('log\_message'))
309. {
310. function log\_message($level = 'error', $message, $php\_error = FALSE)
311. {
312. static $\_log;
314. if (config\_item('log\_threshold') == 0)
315. {
316. return;
317. }
319. $\_log =& load\_class('Log');
320. $\_log->write\_log($level, $message, $php\_error);
321. }
322. }
324. // ------------------------------------------------------------------------
326. /\*\*
327. \* Set HTTP Status Header
328. \*/
329. if ( ! function\_exists('set\_status\_header'))
330. {
331. function set\_status\_header($code = 200, $text = '')
332. {
333. //此函数构造一个响应头。$stati为响应码与其响应说明。
334. $stati = array(
335. 200 => 'OK',
336. 201 => 'Created',
337. 202 => 'Accepted',
338. 203 => 'Non-Authoritative Information',
339. 204 => 'No Content',
340. 205 => 'Reset Content',
341. 206 => 'Partial Content',
343. 300 => 'Multiple Choices',
344. 301 => 'Moved Permanently',
345. 302 => 'Found',
346. 304 => 'Not Modified',
347. 305 => 'Use Proxy',
348. 307 => 'Temporary Redirect',
350. 400 => 'Bad Request',
351. 401 => 'Unauthorized',
352. 403 => 'Forbidden',
353. 404 => 'Not Found',
354. 405 => 'Method Not Allowed',
355. 406 => 'Not Acceptable',
356. 407 => 'Proxy Authentication Required',
357. 408 => 'Request Timeout',
358. 409 => 'Conflict',
359. 410 => 'Gone',
360. 411 => 'Length Required',
361. 412 => 'Precondition Failed',
362. 413 => 'Request Entity Too Large',
363. 414 => 'Request-URI Too Long',
364. 415 => 'Unsupported Media Type',
365. 416 => 'Requested Range Not Satisfiable',
366. 417 => 'Expectation Failed',
368. 500 => 'Internal Server Error',
369. 501 => 'Not Implemented',
370. 502 => 'Bad Gateway',
371. 503 => 'Service Unavailable',
372. 504 => 'Gateway Timeout',
373. 505 => 'HTTP Version Not Supported'
374. );
376. //如果调用此函数本身出错，则发出一个错误。
377. if ($code == '' OR ! is\_numeric($code))
378. {
379. show\_error('Status codes must be numeric', 500);
380. }
382. if (isset($stati[$code]) AND $text == '')
383. {
384. $text = $stati[$code];
385. }
387. //如果$text为空，一般是因为调用此函数时，给的响应码不正确同时又没有写出响应报文信息。
388. if ($text == '')
389. {
390. show\_error('No status text available. Please check your status code number or supply your own message text.', 500);
391. }
393. //取得当前协议。
394. $server\_protocol = (isset($\_SERVER['SERVER\_PROTOCOL'])) ? $\_SERVER['SERVER\_PROTOCOL'] : FALSE;
396. //php\_sapi\_name()方法可以获得PHP与服务器之间的接口类型，
397. //下面是以cgi类型和以服务器模块形式类型的不同发出响应的方式。
398. if (substr(php\_sapi\_name(), 0, 3) == 'cgi')
399. {
400. header("Status: {$code} {$text}", TRUE);
401. }
402. elseif ($server\_protocol == 'HTTP/1.1' OR $server\_protocol == 'HTTP/1.0')
403. {
404. header($server\_protocol." {$code} {$text}", TRUE, $code);
405. }
406. else
407. {
408. header("HTTP/1.1 {$code} {$text}", TRUE, $code);
409. }
410. }
411. }
413. // --------------------------------------------------------------------
415. /\*\*
416. \* Exception Handler
417. \*/
418. if ( ! function\_exists('\_exception\_handler'))
419. {
420. //在CodeIgniter.php中执行set\_error\_handler('\_exception\_handler');后，以后一切非致命(非fatal)错误信息都由它处理。
421. //触发错误的时候，会产生几个参数，错误级别（号），错误信息，错误文件，错误行。
422. function \_exception\_handler($severity, $message, $filepath, $line)
423. {
424. /\*\*
425. \* 有关错误等级的内容可看：http://blog.163.com/wu\_guoqing/blog/static/19653701820127269312682/
426. \* E\_STRICT对于大多数情况来说都是没多大作用的错误提示，这里CI把它屏蔽掉，如果实在要查看，可以查看日志文件。
427. \*/
428. if ($severity == E\_STRICT)
429. {
430. return;
431. }
433. //真正起到错误处理的是Exception组件。
434. $\_error =& load\_class('Exceptions', 'core');
436. /\*
437. \* 注意下面的符号是&而不是&&，php的错误等级的值都是有规律的，例如1,2,4,8...（1,10,100,1000）等等，实际上，php是通过位运算来实现的，
438. \* 使得错误控制更精准。（类似linux的权限控制，rwx）
439. \* 在设置error\_reporting()的时候，可通过E\_XX|E\_YY|E\_ZZ的形式来设置，而判断的时候则通过E\_XX&error\_repoorting()来判断
440. \* E\_XX有没有设置。例如1,10,100,1000相或|，则值为1111,则以后1,10,100,1000中任意一个与1111相&,值都为它本身。
441. \* 而E\_ALL可以看到是除E\_STRICT之外其它等级的“或(|)运算”。个人理解，之所以E\_ALL的值是不同版本有所不同的，是
442. \* 因为有时候会加入新的错误级别，从而导致这个E\_ALL的值也不一样。
443. \*/
444. if (($severity & error\_reporting()) == $severity)
445. {
446. //如果符合则交给Exception组件的show\_php\_error();进行处理。
447. $\_error->show\_php\_error($severity, $message, $filepath, $line);
448. }
450. //下面两行只是根据配置文件判断要不要log错误信息而已。
452. if (config\_item('log\_threshold') == 0)
453. {
454. return;
455. }
457. $\_error->log\_exception($severity, $message, $filepath, $line);
458. }
459. }
461. // --------------------------------------------------------------------
463. /\*\*
464. \* Remove Invisible Characters
465. \*/
466. if ( ! function\_exists('remove\_invisible\_characters'))
467. {
468. function remove\_invisible\_characters($str, $url\_encoded = TRUE)
469. {
470. $non\_displayables = array();
472. // every control character except newline (dec 10)
473. // carriage return (dec 13), and horizontal tab (dec 09)
475. if ($url\_encoded)
476. {
477. $non\_displayables[] = '/%0[0-8bcef]/'; // url encoded 00-08, 11, 12, 14, 15
478. $non\_displayables[] = '/%1[0-9a-f]/'; // url encoded 16-31
479. }
481. $non\_displayables[] = '/[\x00-\x08\x0B\x0C\x0E-\x1F\x7F]+/S'; // 00-08, 11, 12, 14-31, 127
483. do
484. {
485. $str = preg\_replace($non\_displayables, '', $str, -1, $count);
486. }
487. while ($count);
489. return $str;
490. }
491. }
493. // ------------------------------------------------------------------------
495. /\*\*
496. \* Returns HTML escaped variable
497. \*/
498. if ( ! function\_exists('html\_escape'))
499. {
500. function html\_escape($var)
501. {
502. if (is\_array($var))
503. {
504. return array\_map('html\_escape', $var);
505. }
506. else
507. {
508. return htmlspecialchars($var, ENT\_QUOTES, config\_item('charset'));
509. }
510. }
511. }

[CodeIgniter源码分析之CodeIgniter.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之CodeIgniter.php%20–%20Calix的博客_files\CodeIgniter源码分析之CodeIgniter.php%20–%20Calix的博客.html)

1. <?php if ( ! defined('BASEPATH')) exit('No direct script access allowed');
2. /\*\*
3. \* 上面：
4. \* 这个BASEPATH，就是在入口文件(index.php)里面定义的那个BASEPATH～
5. \* 如果没有定义BASEPATH，那么直接退出，下面程序都不执行。其实除了入口文件index.php开头没有这句话之外，所有文件都会有这句话
6. \* 也就是说，所有文件都不能单独运行，一定是index.php在运行过程中把这些文件通
7. \* 过某种方式引进来运行，所以只有入口文件index.php才能被访问。
8. \*
9. \*/
11. /\*\*
12. \* CodeIgniter
13. \*
14. \*/
16. /\*\*
17. \* 弱弱地建议：
18. \* 其实把CodeIgniter.php这个文件的代码运行一次，就是整个CI应用都完成了一次完整的运作流程了。
19. \* 其中会加载一些组件，引入很多外部文件，等等。所以建议在阅读此文件代码的时候，第一遍先阅读它的
20. \* 大概流程，也就是说不必进入相应的组件、函数文件中去。第二遍看的时候才具体看那些函数、组件里面是怎
21. \* 么实现的。当然要看个人需要咯～
22. \*
23. \*/
25. // ------------------------------------------------------------------------
27. /\*\*
28. \* System Initialization File
29. \*/
31. /\*\*
32. \* CodeIgniter Version
33. \*
34. \* @var string
35. \*
36. \*/
37. define('CI\_VERSION', '2.1.2');
39. /\*\*
40. \* CodeIgniter Branch (Core = TRUE, Reactor = FALSE)
41. \*
42. \* @var boolean
43. \*
44. \*/
45. define('CI\_CORE', FALSE);
47. /\*
48. \* ------------------------------------------------------
49. \* Load the global functions
50. \* ------------------------------------------------------
51. \*/
52. //在这里引入了很多全局函数。这些函数在整个应用运行的过程都是非常重要的。
53. //这些函数都是应该最先被引入，起码要先于下面的组件，为什么呢？详见core/Common.php，建议在这个地方先略过。
54. require(BASEPATH.'core/Common.php');
56. /\*
57. \* ------------------------------------------------------
58. \* Load the framework constants
59. \* ------------------------------------------------------
60. \*/
61. //加载配置的常量。这个配置文件里面默认已经有一些和文件有关的常量。
62. //下面这个判断可以看出一开始我们在index.php里面定义的那个ENVIRONMENT的作用之一，如果是定义某个环境，
63. //会调用相应的配置文件，这样就可以使得应用在相应的环境中运行。不仅仅是这个常量的配置文件是这样子，
64. //以后你会发现，其实全部配置文件都是先判断当前环境再引入。
65. //方便切换，只需在index.php里面改一下ENVIRONMENT的值。
66. if (defined('ENVIRONMENT') AND file\_exists(APPPATH.'config/'.ENVIRONMENT.'/constants.php'))
67. {
68. require(APPPATH.'config/'.ENVIRONMENT.'/constants.php');
69. }
70. else
71. {
72. //当然啦，如果压根没有这个环境下的配置文件，就会调用默认的。CI手册上也有说，各种环境下的相同的配置文件，可以直接放在
73. //config/下，而不需要每个环境的目录下都有。
74. require(APPPATH.'config/constants.php');
75. }
77. /\*
78. \* ------------------------------------------------------
79. \* Define a custom error handler so we can log PHP errors
80. \* ------------------------------------------------------
81. \*/
82. //这里是把错误的处理交给\_exception\_handler这个函数来做，是在core/Common.php文件中定义的全局函数。
83. //具体请看core/Common.php。
84. set\_error\_handler('\_exception\_handler');
86. if ( ! is\_php('5.3'))
87. {
88. //这个magic\_quotes就是会自动把那些由$\_GET,$\_POST等传过来的值进行处理，加\。
89. //这个东西最好不要打开，虽然某些时候会帮到忙，不过过滤、转义等处理最后还是手动做好一些。
90. //php5.3以上默认是把这个东西关掉的(linux下/etc/php.ini里面)。这个东西本来就不应该有。
91. @set\_magic\_quotes\_runtime(0); // Kill magic quotes
92. }
94. /\*
95. \* ------------------------------------------------------
96. \* Set the subclass\_prefix
97. \* ------------------------------------------------------
98. \*
99. \*/
100. /\*\*
101. \* 这个subclass\_prefix是什么？
102. \* 这是一个非常棒的东西！！有了它我们就可以轻易扩展CI框架～
103. \* 具体core/Common.php 中的load\_class()
104. \*/
105. if (isset($assign\_to\_config['subclass\_prefix']) AND $assign\_to\_config['subclass\_prefix'] != '')
106. {
107. //这个get\_config($replace)就是从配置文件里面读取信息，这里是读取config/config.php中的配置信息
108. //这个参数$replace的作用是什么呢？就是临时把修改配置文件的意思，注意并没有从改变文件的值，这个改变只是
109. //停留在内存的层面上。
110. //而$assign\_to\_config['xxx'];是在index.php中定义的一个配置信息数组，这个配置数组要优先权要大于配置文件当中的。
111. //所以这个判断的作用是，看看有没有在index.php里面定义了 $assign\_to\_config['subclass\_prefix']，如果有的话，
112. //就那把配置文件中的$config['subclass\_prefix']的值改成这个。
113. get\_config(array('subclass\_prefix' => $assign\_to\_config['subclass\_prefix']));
114. }
116. /\*
117. \* ------------------------------------------------------
118. \* Set a liberal script execution time limit
119. \* 上面这个'liberal'是“慷慨大方的”的意思 -\_-||
120. \* ------------------------------------------------------
121. \*/
122. //这个safe\_mode也是php.ini里面的一个配置项目
123. //具体看：http://blog.163.com/wu\_guoqing/blog/static/196537018201272710925363/
124. //这里主要是设置运行时间限制而已。
125. if (function\_exists("set\_time\_limit") == TRUE AND @ini\_get("safe\_mode") == 0)
126. {
127. @set\_time\_limit(300);
128. }
130. /\*
131. \* ------------------------------------------------------
132. \* Start the timer... tick tock tick tock...
133. \* ------------------------------------------------------
134. \*/
135. //load\_class()是用来加载类的，准确来说，是用来取得某个类的一个单一实例。
136. //像下来的调用就是返回system/core/Benchmark的一个实例。core里面的大都是CI的组件。
137. /\*\*
138. \* Benchmark是基准点的意思。
139. \* 其实这个类的功能很简单，只是用来计算程序运行消耗的时间和内存而已。
140. \* 以后经常在某个地方冒出来句$BM->mark('xxx');这个作用就是记录运行到当前位置的时候的时间点。
141. \* 通过两个时间点，就可以计算出时间了。
142. \*/
143. $BM =& load\_class('Benchmark', 'core');
144. $BM->mark('total\_execution\_time\_start');
145. $BM->mark('loading\_time:\_base\_classes\_start');
147. /\*
148. \* ------------------------------------------------------
149. \* Instantiate the hooks class
150. \* ------------------------------------------------------
151. \*/
152. //取得Hooks组件。
153. /\*\*
154. \* 这个hook也是非常非常棒的一个东西！它可以让我们很好地扩展和改造CI～
155. \* 可以这样理解，一个应用从运行到结束这个期间，CI为我们保留了一些位置，在这些位置上面可以让开发人员放上所谓的
156. \* “钩子”（其实就是一段程序啦！），在应用运行过程中，当运行到有可以放钩子的位置的时候，先检测开发人员有没有
157. \* 实现这里的钩子，如果有就运行它。
158. \* 有些地方甚至可以用自己写的钩子程序替代CI框架本来的程序。
159. \*/
160. $EXT =& load\_class('Hooks', 'core');
162. /\*
163. \* ------------------------------------------------------
164. \* Is there a "pre\_system" hook?
165. \* ------------------------------------------------------
166. \*/
167. //这里就有一个钩子啦。大概意思是：整个应用系统开始动了，这里要不要先让开发人员来一段程序？
168. //如果你定义了pre\_system这个钩子，那么，其实它就是在这个位置运行的。
169. $EXT->\_call\_hook('pre\_system');
171. /\*
172. \* ------------------------------------------------------
173. \* Instantiate the config class
174. \* ------------------------------------------------------
175. \*/
176. //取得配置组件。
177. $CFG =& load\_class('Config', 'core');
179. // Do we have any manually set config items in the index.php file?
180. //如果有在index.php定义配置数组，那么就丢给配置组件CFG，以后就由CFG来保管了配置信息了。
181. if (isset($assign\_to\_config))
182. {
183. $CFG->\_assign\_to\_config($assign\_to\_config);
184. }
186. /\*
187. \* ------------------------------------------------------
188. \* Instantiate the UTF-8 class
189. \* ------------------------------------------------------
190. \*
191. \*
192. \*/
193. //取得UTF－8组件，这里暂时不用管它。
194. $UNI =& load\_class('Utf8', 'core');
196. /\*
197. \* ------------------------------------------------------
198. \* Instantiate the URI class
199. \* ------------------------------------------------------
200. \*/
201. //取得URI组件。
202. $URI =& load\_class('URI', 'core');
204. /\*
205. \* ------------------------------------------------------
206. \* Instantiate the routing class and set the routing
207. \* ------------------------------------------------------
208. \*/
209. //取得URI的好基友RTR
210. $RTR =& load\_class('Router', 'core');
211. //RTR的这个\_set\_routing();其实做了非常多的事情。。详见core/Router.php。非常重要。
212. $RTR->\_set\_routing();
214. // Set any routing overrides that may exist in the main index file
215. //
216. if (isset($routing))
217. {
218. //这个$routing是在index.php入口文件中可以配置的一个数组。这里起到路由覆盖的作用。
219. //index.php里面配置的信息永远都是最优先的。
220. //在这里无论你请求的路由是什么，只要有配置$routing（当然要配对），就会被它重定向。
221. //所以我觉得这句话放在这个地方有点坑，上面\_set\_routing搞了那么久，一下子就被覆盖掉了。
222. $RTR->\_set\_overrides($routing);
223. }
225. /\*
226. \* ------------------------------------------------------
227. \* Instantiate the output class
228. \* ------------------------------------------------------
229. \*/
230. //输出组件。这个输出组件有什么用？输出不是$this->load->view()么？其实它们两个也是好基友。
231. //详见：core/Output.php core/Loader.php
232. $OUT =& load\_class('Output', 'core');
234. /\*
235. \* ------------------------------------------------------
236. \* Is there a valid cache file? If so, we're done...
237. \* ------------------------------------------------------
238. \*/
239. //下面是输出缓存的处理，这里允许我们自己写个hook来取替代CI原来Output类的缓存输出。
240. if ($EXT->\_call\_hook('cache\_override') === FALSE)
241. {
242. if ($OUT->\_display\_cache($CFG, $URI) == TRUE)
243. {
244. exit;//如果可以输出缓存，那么就没有必要再做其它事了。输出结果后直接退出。
245. }
246. }
248. /\*
249. \* -----------------------------------------------------
250. \* Load the security class for xss and csrf support
251. \* -----------------------------------------------------
252. \*/
253. //取得安全组件（安全组件暂时不详讲，因为对于CI一个运作流程来说，它不是必要的。CI的安全处理以后会作为一个新话题来探讨）
254. $SEC =& load\_class('Security', 'core');
256. /\*
257. \* ------------------------------------------------------
258. \* Load the Input class and sanitize globals
259. \* ------------------------------------------------------
260. \*/
261. //取得安全组件的好基友INPUT组件。（主要是结合安全组件作一些输入方面的安全处理，$this->input->post()这些常用的操作都是
262. //由它们两个负责的。）
263. $IN =& load\_class('Input', 'core');
265. /\*
266. \* ------------------------------------------------------
267. \* Load the Language class
268. \* ------------------------------------------------------
269. \*/
270. //语言组件。
271. $LANG =& load\_class('Lang', 'core');
273. /\*
274. \* ------------------------------------------------------
275. \* Load the app controller and local controller
276. \* ------------------------------------------------------
277. \*
278. \*/
279. //引入控制器父类文件。这里和其它组件的引入方式不一样，用load\_class();因为我们最终用到的并不是这个父类，
280. //而是我们自己写在application/controllers/下的某个由uri请求的控制器。
281. require BASEPATH.'core/Controller.php';
283. //定义get\_instance();方法，通过调用CI\_Controller::get\_instance()可以实现单例化，
284. //调用此函数可方便以后直接取得当前应用控制器。
285. function &get\_instance()
286. {
287. return CI\_Controller::get\_instance();
288. }

291. //和其它组件一样，控制器父类同样可以通过前缀的方式进行扩展。
292. if (file\_exists(APPPATH.'core/'.$CFG->config['subclass\_prefix'].'Controller.php'))
293. {
294. require APPPATH.'core/'.$CFG->config['subclass\_prefix'].'Controller.php';
295. }
297. if ( ! file\_exists(APPPATH.'controllers/'.$RTR->fetch\_directory().$RTR->fetch\_class().'.php'))
298. {
299. //其实如果能够进入这里，说明了上面的$RTR->\_set\_routing();在\_validate\_request()的时候一定是在请求默认控制器。
300. //详见：core/Router.php
301. show\_error('Unable to load your default controller. Please make sure the controller specified in your Routes.php file is valid.');
302. }
304. //引入我们最终确认的应用控制器。
305. include(APPPATH.'controllers/'.$RTR->fetch\_directory().$RTR->fetch\_class().'.php');
307. $BM->mark('loading\_time:\_base\_classes\_end');//这里mark一下，说明CI的所需要的基本的类都加载完了。
309. /\*
310. \* ------------------------------------------------------
311. \* Security check
312. \* ------------------------------------------------------
313. \*
314. \* 下面主要进行一些方法上的验证。
315. \* 因为毕竟我们是通过URI直接调用控制器里面的方法的，其实这是个很危险的事情。
316. \* 必须要保证我们原本没想过要通过URI访问的方法不能访问。
317. \*
318. \* CI里面规定以\_下划线开头的方法，一般是作为非公开的方法，即使方法定义为public。
319. \* 其实不仅仅是CI这么做，把非公开的方法名以\_开头，是很好的一种规范。
320. \* 第二个就是父类CI\_Controller里面的方法也是不允许通过URI访问的。
321. \* 如果URI请求这样的方法，那么会作为404处理。
322. \*/
323. $class = $RTR->fetch\_class();
324. $method = $RTR->fetch\_method();
326. if ( ! class\_exists($class)
327. OR strncmp($method, '\_', 1) == 0
328. OR in\_array(strtolower($method), array\_map('strtolower', get\_class\_methods('CI\_Controller')))
329. )
330. {
331. if ( ! empty($RTR->routes['404\_override']))
332. {
333. $x = explode('/', $RTR->routes['404\_override']);
334. $class = $x[0];
335. $method = (isset($x[1]) ? $x[1] : 'index');
336. if ( ! class\_exists($class))
337. {
338. if ( ! file\_exists(APPPATH.'controllers/'.$class.'.php'))
339. {
340. show\_404("{$class}/{$method}");
341. }
343. include\_once(APPPATH.'controllers/'.$class.'.php');
344. }
345. }
346. else
347. {
348. show\_404("{$class}/{$method}");
349. }
350. }
352. /\*
353. \* ------------------------------------------------------
354. \* Is there a "pre\_controller" hook?
355. \* ------------------------------------------------------
356. \*/
357. //这里又一个钩子，这个钩子的位置往往都是在一些特殊的位置的，像这里就是发生在实例化控制器前。
358. $EXT->\_call\_hook('pre\_controller');
360. /\*
361. \* ------------------------------------------------------
362. \* Instantiate the requested controller
363. \* ------------------------------------------------------
364. \*/
365. // Mark a start point so we can benchmark the controller
366. $BM->mark('controller\_execution\_time\_( '.$class.' / '.$method.' )\_start');
368. $CI = new $class();//折腾了这么久，终于实例化我们想要的控制器了。。。。。。。。。。。。。。。。。。。。。
370. /\*
371. \* ------------------------------------------------------
372. \* Is there a "post\_controller\_constructor" hook?
373. \* ------------------------------------------------------
374. \*/
375. $EXT->\_call\_hook('post\_controller\_constructor');
377. /\*
378. \* ------------------------------------------------------
379. \* Call the requested method
380. \* ------------------------------------------------------
381. \*/
382. // Is there a "remap" function? If so, we call it instead
383. if (method\_exists($CI, '\_remap'))
384. {
385. //这个\_remap也是一个非常棒的东西！如果有定义，它会优先被调用，在这个方法里面我们可以根据$method和参数，随意
386. //做任何处理和加工。即使那个$method不存在。
387. $CI->\_remap($method, array\_slice($URI->rsegments, 2));
388. }
389. else
390. {
391. //如果请求的方法不存在，那么就按404处理。
392. if ( ! in\_array(strtolower($method), array\_map('strtolower', get\_class\_methods($CI))))
393. {
394. // Check and see if we are using a 404 override and use it.
395. if ( ! empty($RTR->routes['404\_override']))
396. {
397. $x = explode('/', $RTR->routes['404\_override']);
398. $class = $x[0];
399. $method = (isset($x[1]) ? $x[1] : 'index');
400. if ( ! class\_exists($class))
401. {
402. if ( ! file\_exists(APPPATH.'controllers/'.$class.'.php'))
403. {
404. show\_404("{$class}/{$method}");
405. }
407. include\_once(APPPATH.'controllers/'.$class.'.php');
408. unset($CI);//这里是把原来的控制器删掉，改用我们定义的404重定向的类。
409. $CI = new $class();
410. }
411. }
412. else
413. {
414. show\_404("{$class}/{$method}");
415. }
416. }
418. //终于调用了！！！！！！！！！！！！！！！！！！！！！！！！！！！就在这里。
419. //不过，不是打击你，虽然我们请求的控制器的那个方法被调用了，但是实际上，我们想要的输出并没有完全输出来。
420. //这就是因为$this->load->view();并不是马上输出结果，而是把结果放到缓冲区，然后最后Output类把它冲出来。
421. call\_user\_func\_array(array(&$CI, $method), array\_slice($URI->rsegments, 2));
422. }

425. // Mark a benchmark end point
426. $BM->mark('controller\_execution\_time\_( '.$class.' / '.$method.' )\_end');
428. /\*
429. \* ------------------------------------------------------
430. \* Is there a "post\_controller" hook?
431. \* ------------------------------------------------------
432. \*/
433. $EXT->\_call\_hook('post\_controller');
435. /\*
436. \* ------------------------------------------------------
437. \* Send the final rendered output to the browser
438. \* ------------------------------------------------------
439. \*/
440. if ($EXT->\_call\_hook('display\_override') === FALSE)
441. {
442. $OUT->\_display();//这里，把$this->load->view();里面缓冲的输出结果输出，基本上一个流程总算完成了。详见core/Output.php。
443. }
445. /\*
446. \* ------------------------------------------------------
447. \* Is there a "post\_system" hook?
448. \* ------------------------------------------------------
449. \*/
450. $EXT->\_call\_hook('post\_system');
452. /\*
453. \* ------------------------------------------------------
454. \* Close the DB connection if one exists
455. \* ------------------------------------------------------
456. \*/
457. if (class\_exists('CI\_DB') AND isset($CI->db))
458. {
459. $CI->db->close();//关闭连接。
460. }

[CodeIgniter源码分析之index.php](file:///E:\2.%E6%97%A5%E5%B8%B8\%E6%80%BB%E7%BB%93\php\fine\ci\%E6%BA%90%E7%A0%81%E5%88%86%E6%9E%90\CodeIgniter源码分析之index.php%20–%20Calix的博客_files\CodeIgniter源码分析之index.php%20–%20Calix的博客.html)

1. <?php
3. /\*
4. \*---------------------------------------------------------------
5. \* APPLICATION ENVIRONMENT
6. \*---------------------------------------------------------------
7. \*
8. \*
9. \* //配置项目运行的环境，该配置会影响错误报告的显示和配置文件的读取。
10. \*/
11. define('ENVIRONMENT', 'development');
12. /\*
13. \*---------------------------------------------------------------
14. \* ERROR REPORTING
15. \*---------------------------------------------------------------
16. \*
17. \*/
19. if (defined('ENVIRONMENT'))
20. {
21. switch (ENVIRONMENT)
22. {
23. case 'development':
24. error\_reporting(E\_ALL);
25. break;
27. case 'testing':
28. case 'production':
29. error\_reporting(0);
30. break;
32. default:
33. exit('The application environment is not set correctly.');
34. }
35. }
37. /\*
38. \*---------------------------------------------------------------
39. \* SYSTEM FOLDER NAME
40. \*---------------------------------------------------------------
41. \*
42. \*
43. \* 设置系统目录
44. \*/
45. $system\_path = 'system';
47. /\*
48. \*---------------------------------------------------------------
49. \* APPLICATION FOLDER NAME
50. \*---------------------------------------------------------------
51. \*
52. \*
53. \* 设置应用目录
54. \*/
55. $application\_folder = 'application';
57. /\*
58. \* --------------------------------------------------------------------
59. \* DEFAULT CONTROLLER
60. \* --------------------------------------------------------------------
61. \*
62. \*
63. \*/
65. //下面这个地方设置的$routing会对路由有重定向的作用。详见：core/CodeIginter.php
66. // $routing['directory'] = '';
68. // $routing['controller'] = '';
70. // $routing['function'] = '';
72. /\*
73. \* -------------------------------------------------------------------
74. \* CUSTOM CONFIG VALUES
75. \* ------------------------------------------------------------------
76. \* 下面这里提供一个定义配置信息的地方。其实在index.php里面很多地方都可以设置一些配置，像刚才上面的$routing，
78. \* 而在这里设置的配置信息要优先于在config/目录下设置的配置信息。
79. \*/
80. // $assign\_to\_config['name\_of\_config\_item'] = 'value of config item';
82. // --------------------------------------------------------------------
83. // END OF USER CONFIGURABLE SETTINGS. DO NOT EDIT BELOW THIS LINE
84. // --------------------------------------------------------------------
86. /\*
87. \* ---------------------------------------------------------------
88. \* Resolve the system path for increased reliability
89. \* ---------------------------------------------------------------
90. \*/
92. //这个请参考：http://blog.163.com/wu\_guoqing/blog/static/196537018201272512616394/
93. if (defined('STDIN'))
94. {
95. chdir(dirname(\_\_FILE\_\_));
96. }
98. //计算出$system\_path,即核心文件所在的路径。realpath($path)中的$path必须为存在的路径。
99. if (realpath($system\_path) !== FALSE)
100. {
101. $system\_path = realpath($system\_path).'/';
102. }
104. // 确保以/结尾
105. $system\_path = rtrim($system\_path, '/').'/';
107. // 判断是否为正确的目录
108. if ( ! is\_dir($system\_path))
109. {
110. exit("Your system folder path does not appear to be set correctly. Please open the following file and correct this: ".pathinfo(\_\_FILE\_\_, PATHINFO\_BASENAME));
111. }
113. /\*
114. \* -------------------------------------------------------------------
115. \* 在上面配置好一些文件目录信息后，根据这些目录来定义好一些常量。
116. \* -------------------------------------------------------------------
117. \*/
118. // 这个入口文件的文件名，目前是index.php
119. define('SELF', pathinfo(\_\_FILE\_\_, PATHINFO\_BASENAME));
121. // 文件扩展名
122. define('EXT', '.php');
124. // 统一以/为目录分隔符（windows下是/或\，linux下默认是/）
125. define('BASEPATH', str\_replace("\\", "/", $system\_path));
127. // 前端控制器所在的目录。在CI里面就是这个入口文件。
128. define('FCPATH', str\_replace(SELF, '', \_\_FILE\_\_));
130. // 取得核心文件的目录名，具体做法如下：
131. //trim(BASHPATH,'/'):先把BASHPATH给修剪一下，去掉首尾的‘/’
132. //先后通过strrchr(xxx,'/')：把上述得出来的字符串，截取出以最后一个'/'开头到结尾的一个子字符串。
133. //最后再trim(xxx,'/')，去掉两端的‘/’，实质这里是去掉左边的'/'。
134. define('SYSDIR', trim(strrchr(trim(BASEPATH, '/'), '/'), '/'));
136. // 定义应用文件目录
138. //先看看这个被你配置的目录是不是相对于当前文件，如果是的话，就可以成功定义。
139. if (is\_dir($application\_folder))
140. {
141. define('APPPATH', $application\_folder.'/');
142. }
143. else
144. {
145. //如果不是，则再判断相对于核心文件目录是否存在这个应用目录。如果没有就拉倒了。。
146. if ( ! is\_dir(BASEPATH.$application\_folder.'/'))
147. {
148. exit("Your application folder path does not appear to be set correctly. Please open the following file and correct this: ".SELF);
149. }
151. define('APPPATH', BASEPATH.$application\_folder.'/');
152. }
154. /\*
155. \* --------------------------------------------------------------------
156. \* LOAD THE BOOTSTRAP FILE
157. \* --------------------------------------------------------------------
158. \*
159. \*/
161. //实质上，这个前端控制器兼入口文件仅仅是定义当前项目运行的环境和一些常量，而真正起宏观控制作用的是下面这个CodeIgniter.php。
162. require\_once BASEPATH.'core/CodeIgniter.php';
164. /\*\*
165. \* 总结一下这个文件做了一些什么：
166. \* 第一，先设置好当前项目的运行环境，这里主要是错误报告方面的设置，这个放在了整个项目运行的第一位。
167. \* 第二，再配置好一些目录信息，这些都是一定开发人员可以自定义的东西。然后根据配置目录信息，CI会把一些以后会有用的东西定义为常量
168. \* ，为什么要这样做呢？因为以后会在很多不同地方，例如CI里面的各个组件都会用到路径相关的信息，在这里统一计算并定义，以后方
169. \* 便引用和修改。
170. \* 第三，引入CodeIgniter.php进行工作。
171. \*
172. \* 移步至核心文件目录下的core/CodeIgniter.php： http://calixwu.com/2014/11/codeigniter-yuanmafenxi-codeigniter-php.html
173. \*/