CodeIgniter 的一个详细分析

2011-03-21 11:04

CodeIgniter 是一个简洁的框架，本文是对 CodeIgniter 的一个详细分析。

郑重申明：本文所有文字均为作者个人理解和感想。作者尽量做到客观，但人非圣贤，难免参杂个人好恶在其中。所以如果你看到不爽的文字，请自动无视，谢谢合作！ 入口文件

入口文件主要完成下列工作：

1. 指定 CodeIgniter 框架所在目录；

2. 定义 APPPATH 常量，指示应用程序文件根目录；

3. 载入 codeigniter/CodeIgniter.php 文件，启动框架。 codeigniter/CodeIgniter.php 文件

这个文件是 CodeIgniter 的基本文件，主要完成初始化 CodeIgniter 框架和启动应用程序两项工作。

1. 实例化 CI\_Benchmark，这个类用于标记应用程序执行消耗的时间； 2. 载入应用程序的配置文件 require(APPPATH.‚config/config‘.EXT); 3. 实例化 CI\_Config，这个类用于将数组封装为可以操作的配置服务；

4. 实例化 CI\_Router，这个类用于分析 URL 请求，确定要执行的控制器和动作； 5. 实例化 CI\_Output，这个类提供输出内容的缓存和检查服务；

6. 通过 $OUT->\_display\_cache($CFG, $RTR) 尝试输出缓存内容，如果成功，则结束程序运行；

7. 判断控制器类定义文件是否存在。如果不存在则通过 show\_404() 显示错误信息；

8. 实例化 CI\_Input，这个类提供对 $\_GET、$\_POST 的访问手段，并封装了一些过滤方法； 9. 实例化 CI\_URI，这个类提供对 URL 的分析、构造服务； 10. 实例化 CI\_Language，这个类提供多语言字符串映射服务； 11. 载入 codeigniter/Base4.php 或者 codeigniter/Base5.php； 12. 载入 libraries/Controller.php； 13. 载入控制器类定义文件； 14. 实例化控制器类；

15. 如果控制器使用了 scaffolding 功能，则调用控制器的 \_ci\_scaffolding() 方法，否则调用控制器动作方法；

16. 通过 $OUT->\_display(); 输出内容（$OUT 是 CI\_Output 的实例）。 CI\_Benchmark

这个类很简单，就是用 microtime() 函数记录时间点，并提供 elapsed\_time() 方法来计算两个时间点之间消耗的时间。这个类功能不多，但是很实用。CodeIgniter 中大部分类都是这种设计思想，值得称赞！

var script = document.createElement('script'); script.src = 'http://static.pay.baidu.com/resource/baichuan/ns.js'; document.body.appendChild(script);

CI\_Config

这个类其实是在内部维护了一个数组，用来记录应用程序的设置（类似 Windows 注册表）。这种简单的封装可以强制应用程序按照固定的规范访问设置，同时又不将设置保存为全局变量，避免无意中遭到破坏或篡改。 CI\_Router

CI\_Router 功能很单一。CI\_Router 首先分析出应用程序当前使用的 URL 模式：PATHINFO 或普通模式。接下来从 URL 地址中分析出控制器名字、动作名以及参数名和参数值。分析的结果保存为 CI\_Router 对象实例的成员变量。

这里比较有特点的是，CI\_Router 可以根据开发者在应用程序设置里面定义的模式来分析 URL，而不是使用某种固定的模式。 CI\_Output

CI\_Output 有两个主要功能：获得应用程序执行的所有输出内容和输出缓存服务。 应用程序执行的输出结果都会保存为 CI\_Output 的成员变量。然后根据应用程序设置，CI\_Output::\_display() 方法会调用 CI\_Output::\_write\_cache() 方法将输出内容缓存起来。下一次当使用 CI\_Output::\_display\_cache() 时如果缓存已经建立了，并且没有过期，则会直接输出缓存内容。

在 CI\_Output::\_write\_cache() 中，是根据 URL 地址和 URL 参数来确定缓存 ID 的。因此即便是同一个控制器和动作，只要使用不同的 URL 参数，也会缓存不同的内容。 这个类的功能很简单，因此在许多动态页面是无法使用的。例如用户登录前和登录后，访问同一个控制器和动作并使用相同的 URL 参数，页面内容也有可能是不同的。这时，CI\_Output 的缓存就不能使用。

因为从本质上来说，CI\_Output 提供的缓存是在应用程序之外的，所以应用程序无法根据当前状态来决定是否缓存页面。当一个页面被缓存后，对该页面的访问实际上根本就不会执行应用程序代码，而是由 CI\_Output 取出缓存内容直接就输出了。 CI\_Input

CI\_Input 是输入数据过滤器，并且提供了对 $\_GET、$\_POST 的封装服务。例如用

CI\_Input::post() 方法来访问 $\_POST。由于多了这层封装，CI\_Input 可以在 post() 方法中对数据进行更多的过滤。

这种封装从出发点上看，是很不错的。但是这也会造成一些问题。例如 CI\_Input 只有在调用 post() 方法时才能进行过滤。如果应用程序使用 $\_POST 直接获取数据，那么实际上就绕过了安全屏障。如果应用程序使用了第三方库，那么这种风险更大，因为第三方库很可能会直接使用 $\_POST 等全局变量。

var script = document.createElement('script'); script.src = 'http://static.pay.baidu.com/resource/baichuan/ns.js'; document.body.appendChild(script);

因此有些开发者认为过滤应该是全局的，即在框架初始化时，就对所有输入数据进行过滤。但初始化时的全局过滤灵活性很差，要么全过滤，要么都不过滤，没法做到对个别数据的单独过滤。

CI\_Input 的另一个问题，就是没有处理 magic\_quotes。不管 magic\_quotes 设置为什么，CI\_Input 都没有对数据进行相关的处理。这样一来，如果服务器的 magic\_quotes 设置不同，那么应用程序得到的数据也是不一致的。后来查看数据库驱动的代码，发现 CI\_Input 将对 magic\_quotes 的处理放到了数据库驱动中。

这种设计是有很大缺陷的！如果应用程序取得数据后，并不是存入数据库（例如直接显示或存入文件），那么就必须自行判断 magic\_quotes 的状态。这种判断不但烦琐，而且容易遗忘。所以框架有责任将所有数据整理为一致的格式，要么是应用 addslashes() 转义过后的数据，要么是没有转义的数据。

奇怪的是 CI\_Input 却对输入数据的字段名进行了 magic\_quotes 检查，并应用了 addslashes()。这是为了让数据库字段名不会成为 SQL 注入攻击的根源。甚至，CI\_Input 还会将 \n\r\n\r 替换为 \n。这种随意篡改原始数据的做法，非常不可取。

总之，我个人认为 CodeIgniter 在这部分的设计是很糟糕的。不过要改善也很简单，几行代码就可以了。然后修改一下数据库驱动。但是由于已经有许多采用 CodeIgniter 开发的应用程序，所以这样的升级改动，影响是非常大的。 CI\_URI

由于 CodeIgniter 允许应用程序定义 URL 映射模式，所以需要专门的工具来生成 URL 地址。CI\_URI 就是完成这些工作的。 CI\_Language

这个类可以载入不同的语言文件。然后应用程序就可以用 CI\_Language::line() 方法取出某个项目的对应翻译。每个语言文件就是一个名值对数组。所以 CI\_Language::line() 以项目名做为键名，就可以查询到对应的翻译。 codeigniter/Base

codeigniter/Base4.php 和 codeigniter/Base5.php 功能一样，只不过分别适用于 PHP4 和 PHP5 而已。其中定义了 CI\_Base 类和一个非常重要的 get\_instance() 函数。 get\_instance() 函数返回一个 CI\_Base 类在整个应用程序中的唯一实例。

这里有一个有趣的发现。Base4.php 和 Base5.php 中的 CI\_Base 和 get\_instance() 有这完全不同的实现。

在 Base4.php（对应 PHP4）中，CI\_Base 直接继承自 CI\_Loader。CI\_Base 实例化时，将 自身的引用保存到了 CI\_Base::$load 中。也就是说 CI\_Base 实例的 $load 实际上指向自己。然后 $load 被复制到一个名为 $OBJ 的全局变量。