1. 原码分析
   1. 整体流程
      1. 入口文件

入口文件为index.php，主要功能有：

**1.定义全局常量**

（1）比较重要的路径有：SELF：index.php；FCPATH：根目录；APPPATH：extensions目录；BASEPATH：dayrui目录。

**2.引入核心文件/dayrui/core/CodeIgniter.php**

* + 1. 核心文件

核心文件/dayrui/core/CodeIgniter.php，主要功能有：

1.引入核心文件

引入/dayrui/core/Common.php，在其中又引入/dayrui/libraries/drivers下的App.php、Model.php、Controller.php、View.php以及/models/ContentModel.php文件。

2.加载Hooks类，执行pre\_system钩子。

3.加载Config类，执行\_assign\_to\_config方法，将用户自定义的$assign\_to\_config设置的配置加载到配置类中。

4.加载Router类，执行其\_set\_routing方法，格式化SCRIPT\_NAME、REQUEST\_URI、QUERY\_STRING等，并格式化$\_GET['s']、$\_GET['c']、$\_GET['a']等。

5.加载Output类，执行\_display\_cache方法检测是否有缓存文件，如果有，结束脚本，输出缓存文件。

6.引入/dayrui/core/Controller.php和/dayrui/core/controllers/Common.php类，以及根据之前Router类中的值，引入控制器文件并实例化为$CI。

7.执行$OUT的\_display方法，里面调用$CI中的方法，将结果缓存、输出。

8.关闭数据库等。

* + 1. 目录结构

dayrui/core：应该是改写了的CI框架核心文件。

dayrui/database：数据库驱动。

dayrui/libraries：为核心需要的lib文件，有cache缓存管理类等，其中的drivers中的文件是这个cms系统所用到的基础库文件，如App、Modal等文件。

extensions：主要是一些第三方插件，但config这个目录貌似是CI框架的一些配置。

* 1. 核心
     1. 核心文件

**1.App**

（1）是1个abstract类，但是没有人继承它，里面所有的方法都是static，因此外面都是App::使用。

（2）里面都是一些底层类的方法，如get\_config、get\_site\_id、get\_plugin\_id等。

（3）后面还有几个全局函数url、dr\_url、purl等。

**2.Fn\_base**

1个基础虚类，有\_\_set、\_\_get、\_\_call、\_\_toString等方法。

**3.Controller**

（1）也是1个基础虚类，但也没有人继承它，跟App一样，里面所有的方法都是static。

（2）里面的方法应该都是controller中用到的一些工具方法，主要有：

记录$\_GET中的非法字符check\_Get、记录$\_POST中的非法字符check\_Post等。

获取获取当前项目的根目录的URL的get\_base\_url等方法。

**4.Log**

有2个静态方法write和show，写入和显示日志。

**5.Model**

所有model的基础类，都继承自它。

**6.View**

生成视图的1个类。

* + 1. 配置

**1.引入**

（1）在index.php中引入了/config/config.ini.php，里面是应用程序的配置信息。

（2）在CodeIgniter.php中引入/extensions/config/constants.php，里面应该是框架用到的一些常量。

（3）在CodeIgniter.php中引入Config类，执行\_assign\_to\_config方法，将用户自定义的$assign\_to\_config设置的配置加载到配置类中。

在Config类的构造函数中，使用了get\_config方法，引入了/extensions/config/config.ini.php，里面是框架的一些配置。

* 1. 控制器

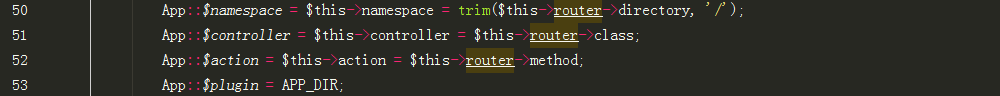
前面的流程可见，最后执行控制器的方法得到输出结果。所有的控制器均继承自/dayrui/core/controllers/Common，进一步继承自/dayrui/core/CI\_Controller。不过CI\_Controller里面没什么内容。但Common中的内容丰富，里面有所有控制器用到的共用方法等。

* + 1. Common

所有控制器均继承该类。主要功能有：

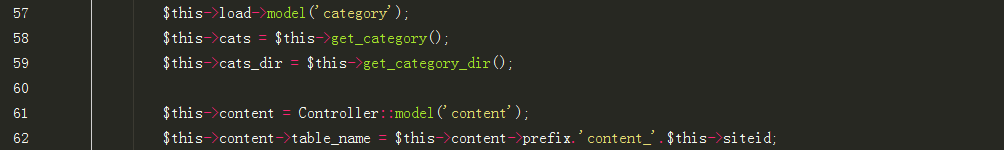
（1）加载一些基础库，如function.php等。

（2）配置MVC底层参数、类到App静态属性中，如：



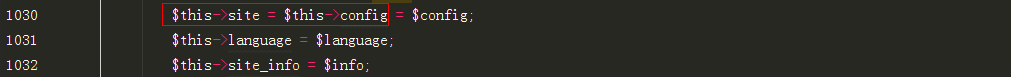


（3）初始化控制器用到的一些应用级别的类库，如：

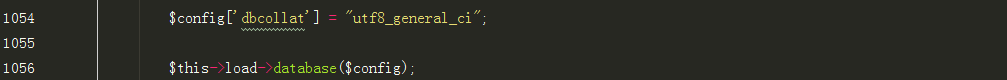




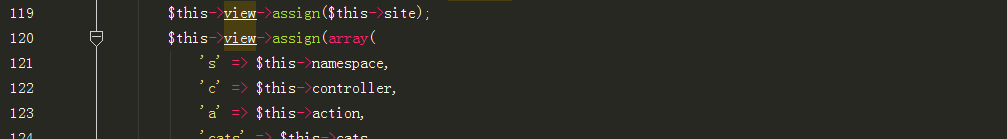
（4）在\_init方法中，加载基础配置。



（5）在\_init方法中，还初始化了数据库。注意这里的$config变量重置了。



（6）设置一部分View用到的参数。



（7）其他共用的操作，如载入会员系统、ip限制、baiduping等。

* + 1. Admin基类

**1.** **构造函数**

主要功能有：

（1）加载用户模块。

（2）检查是否是adminlogin，包括检查mvc的命名空间、名称等，判断是否登录。

（3）使用auth::check检查是否有操作这项功能的权限。

（4）设置view类的用户等参数。

（5）记录操作日志。

**2.** **其他方法**

其他的均为protected方法，被各子类使用，有生成默认菜单的sysMenu方法，投稿审核权限判断verifyPost，指定用户组的操作菜单optionMenu等。

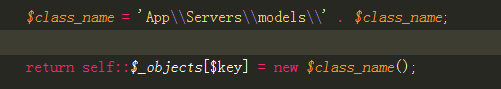
1. 改写
   1. 基本问题

**1.自动加载**

改为composer加载。

http://laravelacademy.org/post/7074.html

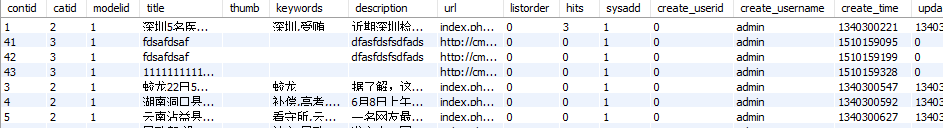
composer dump-autoload

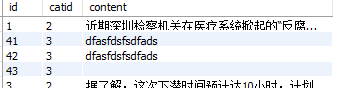


**2.category、content、model三张表的关系**

（1）正文分表

content表不存储文章的正文，正文存储在专门的表中。content表存储除正文外的其他所有信息。





（2）如何通过栏目查找栏目列表需要的正文信息？

这正是content表的具体作用。因为列表只展示desc部分，因此不用查专门表，只查content表即可获取全部所需信息。前提是content表中有catid字段。

（3）如果通过contentid查询一篇文章全部信息，即正文信息和其他信息？

Content表中有1个自增的主键contentid，是content表和专门表建立关系的桥梁。在新增一篇文章时，先获取其他信息插入content表，获取contentid，然后将contentid、catid、正文插入专门表。

但查询时，那么些专门表，如何知道是哪个表？因此需要在插入信息时，将tablename也插入content表，从而找到对应的表。

（4）新增文章如何获取tablename？

需要1个专门的model表，记录专门表的信息。每新增1个专门表，就要将信息记录到model表中。

（5）栏目和model之间的关系？

1个栏目下的文章应该有不同格式，如不同来源的文章格式就可能不同。因此，栏目和model之间应该是1对多关系。不应该是其他关系，要不还得建它们之间的关系表，而且数据没法查。如果有重复，也应该分不同model，保证1对多关系。

如果想单独对1个model设置权限，但showtpl又是一样的，那么可以新建1个model，设置showtpl一样即可。

因此，在新增表时，需要在model表中记录catid、tablename、showtpl。不应该有categorytpl、listtpl，因为只有栏目才应该有这些信息，不会专门展示栏目的1个组成部分。同样，也不应该有setting，只对栏目设置评论、审核等权限。

（6）总结

Category表中不需要有modelid，因为展示栏目首页，列表都用不着查询文章正文。

Model表显然需要有catid，在新增1个表的文章时，要查询model表，从中获取catid、modelid、tablename等，插入content表和正文表。

* 1. 配置管理

controller中的配置在common.php中的init方法中。



* 1. 缓存系统

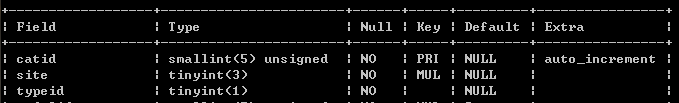
**1.页面缓存**

先全部去掉不管。

**2.Model缓存**

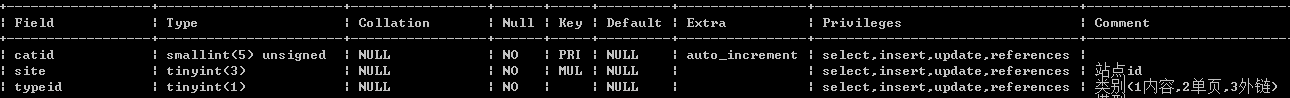
缓存表结构信息和字段信息。表结构信息通过get\_table\_info方法获取。





字段信息通过get\_fields\_type获取。





最后存储格式为：



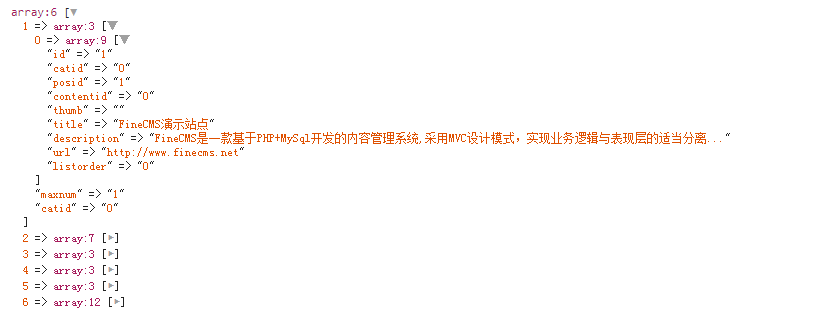
controller缓存

使用cache->set方法设置，均放在data目录下，保存的数据表中的数据。

完全可以根据需要，自己设置缓存。

position

格式为posid为key,对应的data全部信息为数字索引的value,另有maxnum和catid两个值.如下:



tag

格式如下:



需要注意,这里的url生成方式,用到了配置中的SITE\_TAG\_URL。



relatelink



* 1. 多语言
  2. category

删除某个cat，需要同时删除对应的内容表。需要先查找content表中查找modelid，再从model表中查找tablename，然后再从该表删掉对应的数据。同时还要删掉content表中的数据。

cat表和content表是1对多关系。content表和model表是多对1关系，即任何1个content，只属于某个model表id，而model表id可以对应多个content。而cat表和model表目前看来应该也是多对1的关系。如新闻表，可以对应国内、国际等cat。

* 1. model

删除1个model，需要删除content表中该modelid对应的数据，但不用删除表，就像add时没有自动添加table，删除时也就不用自动del，手动删除即可。

删除model跟category也没有关系，category可以继续存在，只是没有了内容而已。

* 1. content

既然内容管理是根据cat树展示，且能根据catid从content表中获取modelid，那么就没有必要在传入modelid。

* 1. tag

**1.添加tag**

tag表只有如下几个字段：



应该是既可以在后台直接添加，也可以在发表文章时添加。

这里的catid为栏目id，即，tag必须和栏目对应。

**2.关联**

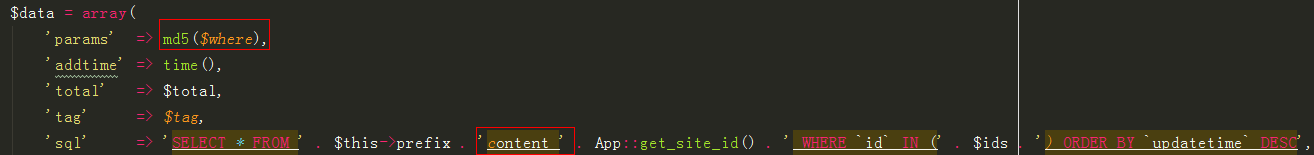
当用户查询tag时，如下：



会从tag\_cache表中首先搜索这个tag对应的数据，如果没有，则调用addData查询并保存。



tag\_cache表保存的内容为：



where条件为：



即根据content的title、keywords、description来搜索tag。



tag\_cache表中sql保存的id就是这里获取的content的id，目的是优化查询，不用以后在模糊查询。

* 1. 文档相关
     1. 审核

status字段，0代表回收，1代表审核通过，2代表拒绝，3代表待审。

* + 1. 相关文档

将相关文档的id保存到relation这个字段中。

* + 1. 积分

将extend表和verify表合并到content表中。

* 1. 改写笔记

cache\_file类最后替换为Cache

* 1. todo

删除cat、model、content时，需要删除对应的position等。还没有做。

position会写入content表中，这样方便查询，非常好。

position表从内容中提取还没有做完。

1. laravel的使用
   1. 命名路由的使用

**1.指定**

使用as。命令路由相当于给路由指定1个别名。



**2.使用**

（1）生成重定向



（2）生成1个url



（3）获取当前路由名



* 1. 跳转

**1.到URL**



（2）重定向到上一个请求的位置



（3）重定向到命名路由





（4）重定向到控制器



（5）带一次性Session数据的重定向

将数据存储到一次性Session中，对于在action之后存储状态信息时特别方便：





* 1. 生成url

（1）直接生成到根目录后的链接。



（2）使用路由名称（as）生成。



（3）根据控制器动作生成，第2个参数可以传参。



（4）当前url



如果需要生产query，可以使用http\_build\_query然后拼接。

* 1. 请求相关

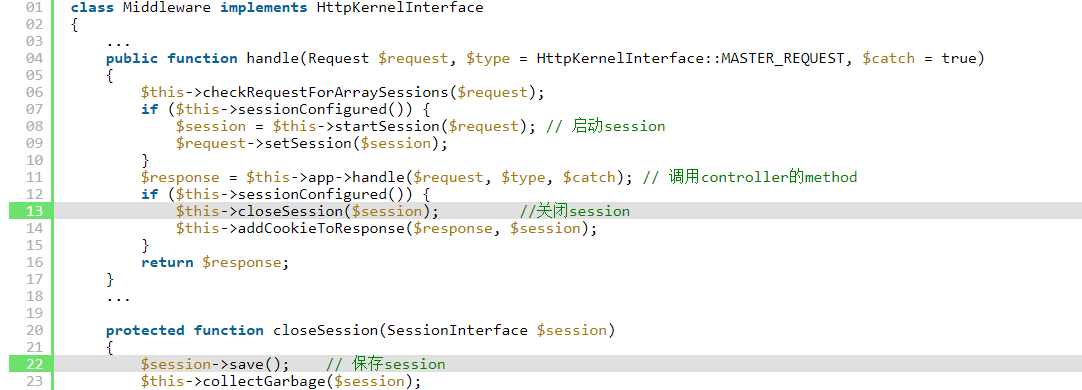
（1）请求方法：method。

（2）是否为ajax：ajax。

（3）获取ip地址：ip。

1. laravel使用
   1. 为什么session无效？

如果代码中有手动的exit，session将无效，原因是put只是将数据保存到Session对象中，最后还需要调用save方法调用原生方法保存。在bootstrap/compiled.php中有：



laravel orm查询出来的时候是1个集合,只能用->，但里面是model，可以用数组方法。

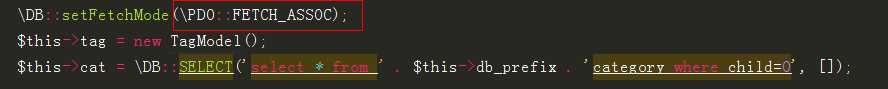
limit使用，使用offset，limit不能接受多个参数。



orderby也不能接受数组，多字段排序需要连写即可。



让DB查询出来的是数组。



composer dump-autoload