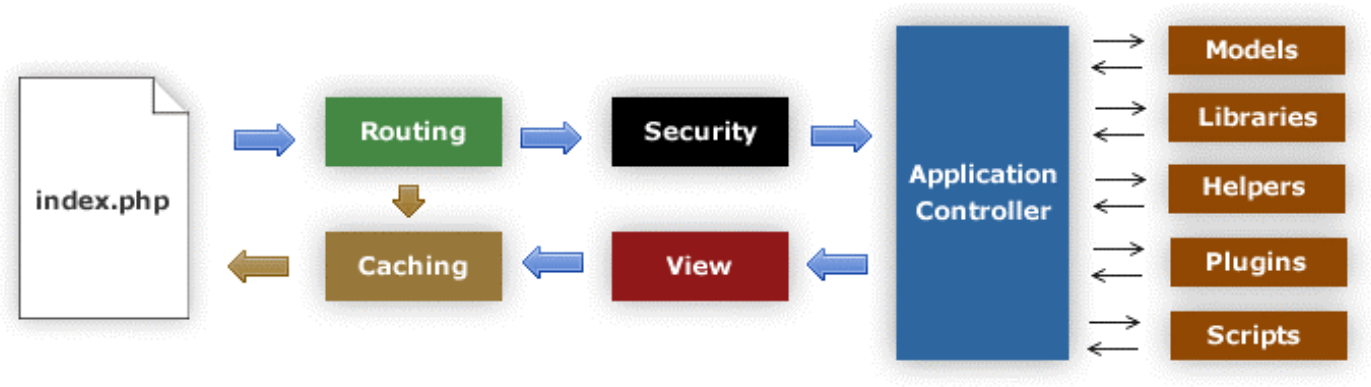
CodeIgniter 架构的使用说明教程

陈德贵 2008-3-7 邮件:degui.chen@qq.com

目录

系统执行的流程图



1. Index.php 服务器前端控制器，初始化运行 CodeIgniter 必须的基本资源
2. 路由器检查 HTTP 请求，为了决定那部分需要执行。
3. 如果缓存文件存在，它直接发送到浏览器，返回正常系统完成。
4. 安全，应用程序控制器被载入之前，HTTP 请求和任何用户提交数据被安全过滤。
5. 控制器加载模型、核心库、辅助文件和其他指定请求资源。

CodeIgniter 默认启动了重定向:

Apche 的配置大约和下面这个差不多. 只是些图片和 JavaScript 目录不重写向

```
<VirtualHost *:80>
    RewriteEngine on
    RewriteCond $1 !^(\/css|\/images|\/js|\/header)
    RewriteRule ^(.*)$ /index.php/$1 [L]
    ServerAdmin degui.chen@qq.com
    DocumentRoot /opt/wwwdata/yum.sg/
    ServerName yum.sg
    ErrorLog yum.sg-error_log
    CustomLog yum.sg-access_log common
</VirtualHost>
```

重定向中的 URL 各部分说明

如: www.your-site.com/class/function/ID

第一个片段代表需要调用的控制器 class。

第二个片段代表需要调用的类中的 function。

第三个片段或更多片段代表传递给控制器的参数，可以是 ID 或任意变量。

CodeIgniter 中的 MVC 架构:

Controller 担当一个在 Model 与 View 之间的中间层, 文件所有位置的目录是: system\application\controllers

Model 扮演你的数据结构。文件所有位置的目录是: system\application\models

View 视图是一个 PHP 的 WEB 页面。文件所有位置的目录是:system\application\views

内部工作流：

1. URL 的控制器初始化

系统初始化后， 直接把控制权交给 Controller ，在 Controller 的初始化时，会把需要的实例化常用的内部类（参考类说明部分），聚合到 Controller 对象中， 作为 Controler 的成员变量。

2. URL 的函数/方法

在 Controler 初始化后,即执行 URL 中第二个片段指定的函数,一般在这里载入 Model 获取所需要的数据。当然在 Controller 初始化时， 也可以载入 Model. 在般在 Controller 中载入的是通用的 Model .而在 函数中则载入专用的 Model

3. 视图显示处理

View 视图是由控制器载入，因为 View 视图是一个 PHP 文件。它的载入函数是在 Controller 函数（即 URL 的函数/方法）内载入。 载入时，系统会把 controller 的所有聚合的成员变量，聚合到 View 中。即：为什么可以在 Controller 中使用的聚合对象，即：View 只是这个函数的内容。这就是可以在 View 中使用的原因：它的实现过程代码大约如下：

```
在 Controller 的 函数内:
$CI =& get Controller ();
foreach (get_object_vars($CI) as $key => $var)
{
    if ( ! isset($this->$key))
    {
        $this->$key =& $CI->$key;
    }
}
```

添加新功能的实例:

添加一个功能的过程:

在控制器目录(system\application\controllers)建立一个控制器文件

(文件名是 URL 的第一个片段的名称, 类名是文件名的首字母大写, 其实类的函数是 URL 的第二个片段的名称)

```
welcome.php
class Welcome extends Controller
{
    function Welcome()
    {
        parent::Controller();
    }

    function index()
    {
        // 这里进行其他处理，如载入类库，进行其他处理。
        $this->load->model(' welcome');
        $this->info = $this-> welcome-> getInfo()
        $this->load->view('welcome_message');
    }
}
```

在模型目录(system\application\models)建立一个模型文件

(文件名是控制器载入的参数名, 类名是文件名的首字母大写)

```
welcome.php
class Welcome extends Model
{
    public function Welcome ()
    {
        parent::Model();
    }

    Public function getInfo()
    {
        return array('degui.chen');
    }
}
```

在视图目录(system\application\views)建立一个视图文件

(文件名也是控制器函数中载入视图时的参数名称)

```
welcome_message.php
print_r($this->->info);
```

一个控制器可以对应多个模型，和多个视图.

主要应用的类使用说明:

载入器

获取当前的控制器

```
$CI =& get_instance();
```

用户载入和实例化类库， 它为用户的应用程序控制器而设计

```
$CI->load->library('email');
$CI->email->emailFunction();
```

载入 Model

```
$CI->load->model('UserModel');
$user_info = $CI->UserModel->GetUserInfo($userid);
```

载入视图文件: 文件的目录在 system/application/views/

直接输出

```
$CI->load->view('viewsTests', array('parameter' => 'value'), false);
直接返回结果
$CI->load->view('viewsTests', array('parameter' => 'value'), true);

载入助手文件(助手文件一般定义的都是些函数)
$CI->load->helper('array');
arrayHlperFuncton();

载入语言文件:
$CI->load->language($file_prefix = 'email', $idiom = 'english');
echo $CI->lang->line('email_must_be_array');

获取数据库对象
function getDatabaseObject()
{
    $CI =& get_instance();
    return $CI->db;
}
```

插入数据到数据库

```
    $table = 'test';
    $data = array('username' => 'username', 'password' => 'password');
    $sql = $database_object->insert_string($table, $data);
    $result = $database_object->simple_query($sql);

更新数据库的数据
    $where = array('id' => $insert_id);
    $table = 'test';
    $data = array('username' => 'username update' . time(), 'password' => 'password' . time());
    $sql = $database_object->update_string($table, $data, $where);
    $result = $database_object->simple_query($sql);
读取数据库的结果
    $sql = 'SELECT * FROM `test` LIMIT 0 , 30';
    $query_result = $database_object->query($sql);
    一行结果
        $result = $query_result->result_object();
        $result = $query_result->result_array();
    多行结果
        $result = $query_result->result('object');
        $result = $query_result->result('array');
```

读取请求的 URL 参数：

```
POST 和 FILES 的数据直接读取 $_POST 和 $_FILES
GET 的参数：
$this->load->library('URI');
//如 URL = http://www.yum.sg/welcome/index/name/value/name1/value1/
$welcome= trim($this->URI->slash_segment(1), '/');
$index    = trim($this->URI->slash_segment(2), '/');
$name     = trim($this->URI->slash_segment(3), '/');
$value    = trim($this->URI->slash_segment(4), '/');
$name1    = trim($this->URI->slash_segment(5), '/');
$value1   = trim($this->URI->slash_segment(6), '/');
```

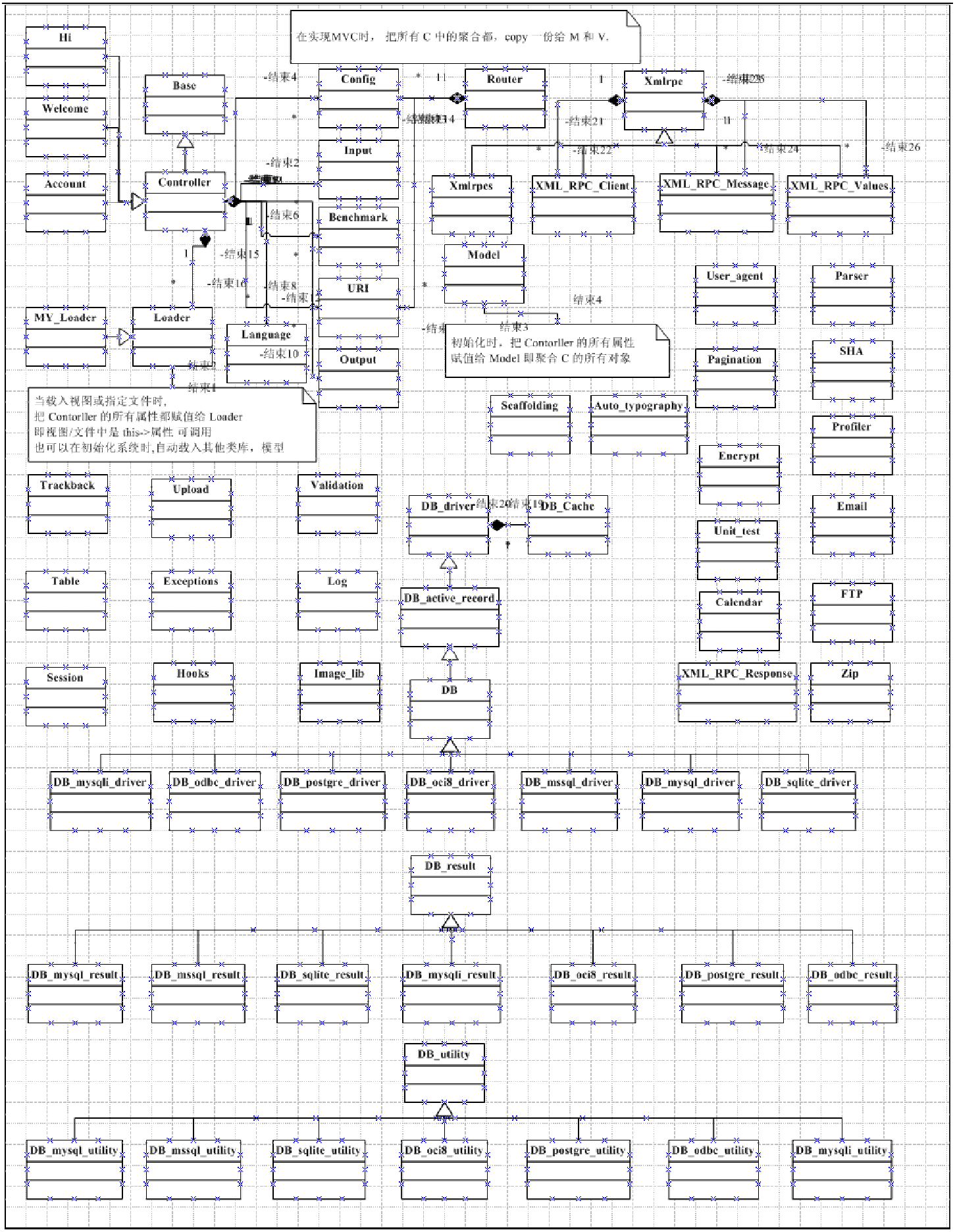
架构中的类的职责说明：

```
Auto_typography         自动印刷类；格式化字符串，格式化新行。
CI_Base                 实现单态函数， 为了方便获取控制器对象。
CI_Benchmark        标准检查:二个标志之间共用的时间 和 内存使用量
CI_Calendar             动态创建日历
CI_Config           提供一种方法获取配置参数，可以是默认是配置文件，也可以是自定义的配置文件。
CI_Email                电子邮件处理类。
CI_Encrypt          加密、解密类。
CI_Exceptions           异常处理类
CI_FTP                  FTP 处理类
CI_Hooks            嵌入和修改架构内部工作方式，不修改核心文件。CodeIgniter 运行它遵循一个指定执行过程。
CI_Image_lib            图片处理类。如水印、调整大小
CI_Input                请求输入类，为了过滤请求数据的安全，和提供对请求数据库的其他处理
CI_Language             目标是国际化，获取语言文件和语言文件的一行内容。
CI_Loader           装载类，负责载入类库（类文件）、视图文件、助手文件、插入文件和自定义文件且实例化它。
CI_Log                  日志处理类， 记录日志
```

CI_Output	输出类，是为了发送最终的页面结果到浏览器，它也为缓存页面负责。
CI_Pagination	分页类。
CI_Parser	剖析类，把标记替换成指定内容。
CI_Profiler	性能统计/调试类，如请求的 SQL 条数。
CI_Router	解释 URI，且决定运行过程的安排。
CI_Session	会话处理类。
CI_SHA	SHA1 编码类
CI_Table	HTML 表格创建类
CI_Trackback	引用通告处理类。
CI_Unit_test	单元测试类
CI_Upload	文件上载处理类
CI_URI	URI 类，解释 URI，且决定运行过程的安排。
CI_User_agent	用户代理类，识别用户的平台、浏览器、机器在人。
CI_Validation	有效性验证类
CI_Xmlrpc	XML-RPC 请求处理类
CI_Xmlrpcs	XML-RPC 服务器类
CI_Zip	Zip 压缩类
Controller	应用程序控制器父类，模型和视图都由它分配聚合的对象。
Model	应用程序模型父类.
Scaffolding	脚手架, 提供了一套在开发过程中快速方便的方法来添加，修改或删除数据库中的信息。
Welcome	应用程序控制器子类的一个实现，测试用的。
XML_RPC_Client	XML-RPC 客户端
XML_RPC_Message	XML-RPC 消息类
XML_RPC_Response	XML-RPC 响应类
XML_RPC_Values	XML-RPC 变量值类
CI_DB_active_record	数据库有效记录类，允许你使用最小的脚本读取、插入、更新你的数据库。
CI_DB_Cache	数据库缓存类，缓存你请求的结果，减小负载。
CI_DB_driver	数据库驱动器.(父类)
CI_DB_result	数据库结果处理类，(父类)
CI_DB_utility	包含有效的函数，帮助你管理你的数据库。(父类)
CI_DB_mssql_driver	数据库驱动器.(子类)
CI_DB_mssql_result	数据库结果处理类，(子类)
CI_DB_mssql_utility	包含有效的函数，帮助你管理你的数据库。(子类)
CI_DB_mysql_driver	数据库驱动器.(子类)
CI_DB_mysql_result	数据库结果处理类，(子类)
CI_DB_mysql_utility	包含有效的函数，帮助你管理你的数据库。(子类)
CI_DB_mysqli_driver	数据库驱动器.(子类)
CI_DB_mysqli_result	数据库结果处理类，(子类)
CI_DB_mysqli_utility	包含有效的函数，帮助你管理你的数据库。(子类)
CI_DB_oci8_driver	数据库驱动器.(子类)
CI_DB_oci8_result	数据库结果处理类，(子类)
CI_DB_oci8_utility	包含有效的函数，帮助你管理你的数据库。(子类)
CI_DB_odbc_driver	数据库驱动器.(子类)
CI_DB_odbc_result	数据库结果处理类，(子类)
CI_DB_odbc_utility	包含有效的函数，帮助你管理你的数据库。(子类)
CI_DB_postgre_driver	数据库驱动器.(子类)
CI_DB_postgre_result	数据库结果处理类，(子类)
CI_DB_postgre_utility	包含有效的函数，帮助你管理你的数据库。(子类)
CI_DB_sqlite_driver	数据库驱动器.(子类)
CI_DB_sqlite_result	数据库结果处理类，(子类)
CI_DB_sqlite_utility	包含有效的函数，帮助你管理你的数据库。(子类)

架构的静态结构图:

架构层的概念: 控制器、模型、视图、载入者、ＸＭＬ－ＲＰＣ、输入、输出、缓存、数据库驱动、数据库查询结果、语言、配置、有效性验证… 辅助性类(如日志、电子邮件等)。


在实现MVC时， 把所有 C 中的聚合都，copy 一份给 M 和 V。
Hi
Welcome
Account
Base
Controller
Config
Router
Xmlrpc
Input
Benchmark
URI
Output
Language
Loader
MY_Loader
Xmlrpcs
XML_RPC_Client
XML_RPC_Message
XML_RPC_Values
Model
初始化时，把 Contorller 的所有属性
赋值给 Model 即聚合 C 的所有对象
当载入视图或指定文件时，
把 Contorller 的所有属性都赋值给 Loader
即视图/文件中是 this->属性 可调用
也可以在初始化系统时,自动载入其他类库，模型
User_agent
Parser
Pagination
SHA
Scaffolding
Auto_typography
Profiler
Encrypt
Email
Unit_test
FTP
Calendar
Zip
XML_RPC_Response
Trackback
Upload
Validation
DB_driver
DB_Cache
Table
Exceptions
Log
DB_active_record
Session
Hooks
Image_lib
DB
DB_mysqli_driver
DB_odbc_driver
DB_postgre_driver
DB_oci8_driver
DB_mssql_driver
DB_mysql_driver
DB_sqlite_driver
DB_result
DB_mysql_result
DB_mssql_result
DB_sqlite_result
DB_mysqli_result
DB_oci8_result
DB_postgre_result
DB_odbc_result
DB_utility
DB_mysql_utility
DB_mssql_utility
DB_sqlite_utility
DB_oci8_utility
DB_postgre_utility
DB_odbc_utility
DB_mysqli_utility