Lge(Lightweight Glue Engine)框架使用文档

1. 简要介绍

Lge 是一款轻量级的组件式框架，框架核心代码文件大小总共不足 800KB，采用面向对象设计以及 MMVC 开发模式，提供了 WEB 开发中常用的功能模块，并易于扩展。

2. 框架特点

- 轻量级，核心代码不足 800K；
- 逻辑简单，结构清晰，运行高效；
- 易于集成到其他框架，并互不影响；

3. 设计理念

3.1. 五层架构

- 数据表现层
- 访问控制层
- 业务逻辑层
- 数据访问层
- 数据持久层

数据表现层：最顶层，主要向用户展示数据，与用户直接交互；
访问控制层：程序结构中的最顶层，根据用户不同的业务调用并初始化对应的资源；
业务逻辑层：业务逻辑层用于处理用户提交的各种操作请求，是整个系统中处理最频繁的一个层；
数据访问层：提供抽象的模块进行数据封装，主要是数据库操作的封装，直接与底层数据库交互；
数据持久层：数据存储的最底层，物理层，由关系数据库以文件的形式保存数据并提供检索功能；

3.2. 设计模式

3.2.1. MMVC

框架基于 MMVC(Model, Module, View, Controller)的设计模式。

- View

属于数据表现层，即视图，用于呈现内容给用户(也就是将程序运行的结果返回给浏览器显示)。例如商品列表页面、后台登录页面。

- Controller

属于访问控制层，即控制器，用于接收用户输入(例如通过浏览器发起的请求)，然后调用模块(Module)对输入数据进行业务逻辑处理并获得处理结果。最后将结果传递到视图(View)，从而让用户能够看到自己操作的结果。例如用户点击删除文章按钮后，控制器调用操作文章的模型，删除掉指定文章，最后通过视图显示成功删除文章的提示信息。

- Module

属于业务逻辑层，即模块，提供项目中对于不同业务的程序封装，为其他模块提供 API 调用接口，并实现业务逻辑的可复用性。

- Model

属于数据访问层，即模型，用于封装管理底层数据。

经过这样简单的分离，我们就把应用程序操作数据的代码(绝大部分 Web 应用程序都是对数据进行操作)和处理用户输入输出的代码分离开来了。
这种分离有许多好处：

✓ 解耦合，清晰地将应用程序分隔为独立的部分；
✓ 业务逻辑的代码能够很方便地在多处重复使用；
✓ 方便开发人员分工协作，提高开发及维护效率；

✓ 可以方便开发人员对应用程序各个部分的代码进行测试；

3.2.2. 单例模式

从执行效率方面考虑，框架很多地方都采用了单例模式，即在同一个请求执行流程中，一种资源只初始化一个操作对象，这个对象在整个的执行流程中被使用。例如，数据库操作对象，同样的数据库资源在同一个请求中只实例化一个操作对象，如果每次数据库操作或者每个模型初始化时都实例化一个数据库操作对象，那么想必这是对资源极大的浪费。

3.2.3. 工厂模式

框架的底层对象都使用了工厂模式进行了封装，并提供了外部接口在需要的地方可直接进行 API 调用并获取需要的资源操作对象。例如：数据库对象、模板对象、Cookie 对象等。

3.3. 子系统/分站

一个项目包含许多分站，封装逻辑比较复杂的，业务功能比较独立的部分，在 LGE 框架的概念里，我们称之为“子系统”，存放在项目源代码根目录的/system 目录下。例如：一个网站的前端站点与后端管理后台可以是两个不同的子系统，项目的定时执行脚本管理功能、项目的微服务功能、用户中心管理后台、网站的 PC 网站和移动端网站也都可以是不同的子系统。

子系统之间业务关联性比较强，需要调用相同的模型(Model)或者模块(Module)来实现某一些功能，但是业务流程太重又需要单独解耦出来(View、Controller)。需要注意的是，子系统有独立的配置文件、控制器、模板文件、自定义类以及第三方类库，子系统之间不能相互调用内部资源，能够相互调用的是全局定义的资源(如全局的配置文件、模型、模块、自定义类和第三方类库)。

4. 文件/目录及命名

4.1. 文件/目录说明

文件/目录	说明
Lge	项目根目录
├── doc	项目文档目录
├── log	项目日志目录
└── src	项目源码目录
├── _cfg	配置文件目录
│ └── _example	配置文件实例文件，用于版本控制中
│ ├── config.inc.php	变量配置文件，不加入版本控制
│ └── const.inc.php	常量配置文件，不加入版本控制
├── _frm	框架目录
│ ├── core	框架核心文件目录
│ ├──library	框架提供类库目录
│ ├──thirdparty	第三方类库目录，不属于框架类库，由第三方提供的类
│ └──common.inc.php	框架包含文件，使用框架时包含该文件到项目中即可
├── _inc	项目类库目录
│ ├── class	用户自定义类存放目录，与项目相关
│ ├──library	第三方服务端插件类库或者系统独立的类库存放目录
│ ├── model	模型
│ └── module	模块
├── cache	缓存或者临时文件存放目录

├─── static	静态文件和前端相关的文件存放目录
│ ├─── html	静态文件
│ ├─── plugin	第三方插件
│ └─── resource	前端资源文件
├─── system	系统目录，下面每一个目录表示一个子系统，用于区分项目的不同部分
│ ├───default	默认子系统
│ │ ├─── _cfg	配置文件目录
│ │ ├─── _ctl	控制器目录
│ │ ├─── _inc	包含文件目录，如类定义文件
│ │ └─── template	模板存放目录
│ ├───admin	(示例)后台管理子系统
│ └─── service	(示例)微服务管理子系统
└─── upload	上传文件存放目录
├─── file	文件或文档
├─── image	图片
└─── media	多媒体

4.2. 文件/目录命名

- 带下划线'_'开头的目录，是项目的核心文件存放目录，只用于被引用或者被其他文件所包含，不能直接用于外部访问(例如不能 http 直接访问)。
- 类库目录和类文件采用英文首字母大写，中间不能夹带下划线(下划线在类自动加载中用于文件寻址)。其他文件和目录统一采用驼峰形式并尽可能采用小写。
- 需要注意的是，类定义文件统一采用英文首字母大写，并且以.class.php 结尾，例如：User.class.php，Login.class.php。

5. 框架文件说明

文件/目录	说明
_frm	框架目录
├─── core	框架核心组件及类文件
│ ├─── component	组件目录
│ │ ├─── FastTpl	模板管理类
│ │ ├─── Config.class.php	配置管理类
│ │ ├─── Cookie.class.php	COOKIE 管理类
│ │ ├─── Database.class.php	数据库管理类
│ │ ├─── Data.class.php	全局变量管理类
│ │ ├─── Debugger.class.php	调试类
│ │ ├─── Instance.class.php	单例管理类
│ │ ├─── Logger.class.php	日志管理类
│ │ └─── Router.class.php	路由管理类
│ ├─── controller	控制器基类目录
│ │ └─── BaseController.class.php	控制器基类
│ ├─── model	模型基类目录
│ │ ├─── BaseModel.class.php	基础模型基类
│ │ └─── BaseModelTable.class.php	基于单表的模型基类
│ ├─── module	模块基类目录

│ │ └─── BaseModule.class.php	模块基类
│ ├─── view	视图基类目录
│ │ └─── Template.class.php	视图封装类
│ ├─── Base.class.php	框架基类，所有的框架核心组件都继承于该类
│ ├─── Core.class.php	框架执行流程引导类
│ ├─── Core.func.php	框架函数定义文件
│ └─── Core.inc.php	框架核心包含文件，用于指定框架运行需要包含的类定义文件以及包含顺序
├─── library	框架类库，用于提供可选择的功能类库
│ ├─── Cache	缓存类目录
│ │ └─── Memcache.class.php	Memcache 管理类
│ ├─── Image	图片处理类目录
│ │ ├─── Utility.class.php	图片工具类，提供常用的图片处理方法
│ │ └─── Validator.class.php	验证图片生成类
│ ├─── Network	网络工具类目录
│ │ ├─── Ftp.class.php	FTP
│ │ └─── Http.class.php	HTTP
│ ├─── NoSQL	NoSQL 类存放目录，目前只有 Redis，但是不需要单独封装，因此该目录为空
│ ├─── ConsoleOption.class.php	命令行参数解析类
│ ├─── FileSys.class.php	文件系统封装类
│ ├─── IpHandler.class.php	IP 工具类
│ ├─── Page.class.php	分页类
│ ├─── Redirecter.class.php	页面跳转控制类
│ ├─── Request.class.php	为简化请求参数处理并获取的类
│ ├─── Response.class.php	返回数据封装类(主要是封装数据格式)
│ ├─── Tree.class.php	树形工具类
│ └─── XmlParser.class.php	XML 解析生成类
├─── thirdparty	第三方类库
└─── common.inc.php	框架包含文件

6. 框架核心功能

6.1. 数据库抽象类

Lge 的 PHP 数据库抽象类基于 PDO 扩展，支持所有主流的数据库管理系统，包括 MySQL, SQLite, MSSQL, PostgreSQL, Oracle 等等，并且天然支持主从读写分离的功能。这些功能都可以通过配置文件简单地配置即可使用。

另外需要说明的是，抽象类的各种封装方法的核心是 PDO 的“预处理”功能，也就是禁止 SQL 使用字符串拼接。“预处理”方式更安全，能有效地防止 SQL 注入。但是 LGE 框架从灵活性以及开发效率上考虑，在数据库的操作方法中同时支持字符串以及数组(预处理)参数，即不强行禁止使用 SQL 拼接方式，但是请在使用的时候小心谨慎。

普通的数据库配置例如：

```
/**
 * 默认数据库配置项.
 */
'default' => array(
    'host'     => '127.0.0.1', // 主机地址(使用 IP 防止 DNS 解析)
    'user'     => 'root',      // 账号
    'pass'     => '',          // 密码
    'port'     => '3306',      // 数据库端口
    'type'     => 'mysql',     // 数据库类型 mysql|pgsql|sqlite
```

```
    'charset'  => 'utf8',      // 数据库编码
    'database' => '',          // 数据库名称
    'linkinfo' => '',          // 可自定义 PDO 数据库连接信息
),
```

其中 linkinfo 配置项可以支持用户自定义的连接(主要是针对于框架不支持的数据库类型)，当该项不为空时，框架将忽略其他连接配置，并以该信息来进行数据库连接，该参数为 PDO 连接数据库的参数(具体参见 PHP-PDO 连接数据库参数形式)，例如自定义 SQLServer 的链接：

```
sqlsrv:Server=127.0.0.1;Database=Test
```

主从读写分离的配置如下：

```
/**
 * 天然支持主从复制模式，当配置项中包含 master 和 slave 字段时，数据库操作自动切换为主从模式，不会读取该配置项内的其他配置。程序在执行数据库操作时会判断优先级，优先级计算方式：配置项值/总配置项值.
 */
'master_slave' => array(
    'master'  => array(
        array(
            'host'     => '127.0.0.1',
            'user'     => 'root',
            'pass'     => '',
            'port'     => '3306',
            'type'     => 'mysql',
            'charset'  => 'utf8',
            'database' => 'test',
            'priority' => 100,
            'linkinfo' => '',
        ),
        array(
            'host'     => '127.0.0.1',
            'user'     => 'root',
            'pass'     => '',
            'port'     => '3306',
            'type'     => 'mysql',
            'charset'  => 'utf8',
            'database' => 'test',
            'priority' => 100,
            'linkinfo' => '',
        ),
    ),
    'slave'   => array(
        array(
            'host'     => '127.0.0.1',
            'user'     => 'root',
            'pass'     => '',
            'port'     => '3306',
            'type'     => 'mysql',
            'charset'  => 'utf8',
            'database' => 'test',
            'priority' => 100,
            'linkinfo' => '',
        ),
        array(
            'host'     => '127.0.0.1',
            'user'     => 'root',
            'pass'     => '',
            'port'     => '3306',
            'type'     => 'mysql',
            'charset'  => 'utf8',
            'database' => 'test',
            'priority' => 100,
            'linkinfo' => '',
        ),
    ),
),
```

使用方法详见 demo: /system/demo/_ctl/Database.class.php

6.2. 执行流程引导

6.2.1. 框架执行流程

框架的执行流程从入口文件开始(例如 index.php)，加载核心的框架文件以及配置文件之后，由框架的 Core.class.php 文件引导整个请求的执行流程，详情可以自行查看这两个文件。

/index.php
/_frm/core/Core.class.php

6.2.2. 业务执行流程

业务执行流程是框架执行流程的一部分，当控制器被初始化后，执行流程将会被引导到特定的控制器方法中，由该方法根据特定的业务需要调用对应的资源执行请求。

6.3. 路由解析功能

6.3.1. 关于请求寻址

在执行流程引导部分，我们可以了解到，一个请求从进入到执行，需要确定 3 个参数，一个是请求属于哪个子系统，第二个是请求需要转交给哪个控制器，第三个是请求需要控制器的哪个方法来处理。

框架对此默认的处理是，子系统名通过__s 参数识别，控制器名通过__c 参数识别，方法名通过__m 参数识别。并且，在入口文件中，用户可以给流程引导类(Core)传递自定义的子系统($sysDir)、控制器($ctl)以及方法($act)参数，用于自定义处理请求参数。

6.3.2. Web Server 的伪静态配置

如果需要框架支持伪静态功能，建议的是在 Web Server 端配置伪静态，以支持框架对于 URI 的解析。

Nginx 的伪静态配置如下(参考)：

```
server {
    listen 80;
    root   项目源代码根目录绝对路径;
    index  index.html index.htm index.php;
    server_name 域名;

    location / {
        try_files $uri /index.php?$query_string;
    }

    location ~ \.php$ {
        include     snippets/fastcgi-php.conf;
        fastcgi_pass unix:/var/run/php5-fpm.sock;
    }
}
```

6.3.3. 路由规则

路由规则是由一系列的正则表达式按照优先级从上往下的顺序进行匹配替换(路由配置比较复杂的是正则表达式的正确书写)。

每个子系统有特定的路由规则，并且相互之间互不影响。路由规则由子系统中的配置文件 router.inc.php 文件指定，如果该文件中的规则不为空，那么将会启用路由规则解析功能。路由规则包含两种：URI 和 URL。

6.3.3.1. URI

URI 规则用于解析请求参数，按照一定的正则表达式匹配，并替换为 Lge 能够解析的参数形式(__s, __c, __m)。注意改部分的正则表达式只会替换 URI 中不包含 QueryString 的部分, QueryString 将不会进入匹配替换判断。

6.3.3.2. URL

URL 规则是将页面渲染输出的内容进行进一步加工处理，按照配置文件的规则优先级从上往下进行正则匹配，替换为伪静态的形式。

6.3.4. 默认规则

可参考子系统配置目录下_example/router.inc.php 文件。

```
array(
    /**
     * URI 解析规则，用于将前端 URI 转换为内部可识别的 GET 变量(主要用于 SEO 或者请求转发到特定的控制器中)。
     */
    'uri' => array(
        // 示例：http://xxx.xxx.xxx/user/list/?type=1&page=2 =>
http://xxx.xxx.xxx/?__c=user&__a=list&type=1&page=2
```

```
        '/\/([\w\.\-]+)\/([\w\.\-]+)[\/\?]*/'         => '__c=$1&__a=$2',
        '/\/([\w\.\-]+)[\/\?]*/'                      => '__c=$1&__a=index',
    ),

    /**
     * 连接转换规则，用于将页面特定规则的连接转换为伪静态连接形式(主要用于 SEO)。
     */
    'url' => array(
        // 示例: http://xxx.xxx.xxx/?__c=user&__a=list&type=1&page=2 =>
http://xxx.xxx.xxx/user/list/?type=1&page=2
        '/\/*(\w+\.php){0,1}\?\_\_s=(\w+)\&\_\_c=(\w+)\&\_\_a=(\w+)[\&]*/' => '/$3/$4/?__s=$2&',
        '/\/*(\w+\.php){0,1}\?\_\_c=(\w+)\&\_\_a=(\w+)[\&]*/'              => '/$2/$3/?',
        '/\/*(\w+\.php){0,1}\?\_\_c=(\w+)[\&]*/'                           => '/$2/?',
    ),
);
```

6.4. 单例对象工厂

框架提供的核心对象都是通过单例模式提供，并由/_frm/core/component/Instance.class.php 封装，具体请查看该文件。

6.5. 全局变量封装

全局变量由两个类进行封装，以便能更方便地管理和维护这些全局变量。

一个是基类/_frm/core/Base.class.php，主要通过魔法成员变量封装 PHP 全局的环境变量如：$_GET, $_POST, $_SESSION 等等，具体请查看该基类文件。

一个是/_frm/core/component/Data.class.php，所有用户自定义的全局变量由 Data 类统一进行管理，包括全局的配置文件变量，框架核心对象等都在此进行注册。在同一个进程中，Data 类中注册的同一个 key 的变量只存在一个，通过 Data::set 写入，通过 Data::get 获取，并且可以通过&Data::get 方式获取变量的引用，减少变量复制，因此能更好地提高运行效率。

6.6. 配置文件封装

配置文件通过/_frm/core/component/Config.class.php 进行封装管理，Config 类只有一个方法，指定配置文件名，通过 Config::get 方法获取配置文件内容。

6.7. COOKIE 功能封装

框架对 cookie 进行了封装管理，并提供了基础的 cookie 加密解密功能，具体请查看文件 /_frm/core/component/Cookie.class.php。

6.8. 轻量级模板引擎

框架使用轻量级的模板引擎 FastTpl，文件目录在/_frm/core/component/FastTpl/。

Demo 脚本路径: /system/demo/_ctl/template/。

6.8.1. 7 个标签，1 个类，带有文件缓存功能，速度特别快

模板标签存在的目的是为解决前后端分工，或者模板编写者不熟悉 PHP 代码，或者限制模板中 PHP 代码权限(PHP 代码也只能实现视图功能，不能涉及任何的业务逻辑或者对底层数据操作)的情况。

1) foreach

   {foreach from=$array key=$key item=$item} {/foreach }

2) for

   {for name=$i min=0 max=15 step=2}{/for}

3) if...elseif...else

   {if $value == xxx || $value = xxx && ($value != xxx || $value == xxx)}

   {else if $value == xxx}

   {else}

   {/if}

4) 变量显示

{$value}

5) 变量赋值

{$value=xxx}

6) 模板加载

{include file.tpl }

7) 注释

{*注释*}

6.8.2. PHP 代码混合支持

<?php PHP 代码，只有在设置允许的条件下才能使用 ?>

如果模板编写者熟悉 PHP 代码，或者前后端是同一位开发者，那么这种情况下可以使用 PHP 代码混合。但需要注意的是，过多的 PHP 代码会造成模板文件的混乱，甚至造成模板文件中产生结构混乱(如模板文件中出现业务逻辑代码，底层数据操作等等)。

6.8.3. 模板引擎插件

FastTpl 支持插件，默认的自带插件存放在/_frm/core/component/FastTpl/plugin 目录，使用示例请参考 demo。由于 FastTpl 也支持 PHP 代码混合，其实用插件的地方也可以通过 PHP 代码来实现，但是过多的 PHP 代码混合会使模板文件混乱不堪。插件存在的目的就是 PHP 代码混合模式关闭的情况下，在这种情况下模板的编写也更加安全，模板代码更加简洁，前后端分工也更加清晰。

6.8.4. 纯 PHP 代码模板模式

FastTpl 支持纯 PHP 代码模板的模式，在这种模式下，7 个标签以及文件缓存功能将会失效，模板引擎将把模板文件当做纯 PHP 文件处理。在这种模式下，模板引擎的执行效率将会更高。

6.9. 类自动加载

6.9.1. Module_* 模块类自动加载

在项目的任何地方调用 Module_*模块类将会寻址到/_inc/module 目录下指定类名的文件，例如：Module_User 类名将会被自动寻址到/_inc/module/User.class.php 文件，并且该文件会被系统自动加载一次。

6.9.2. Modle_* 模型类自动加载

同上，模型类将会寻址到/_inc/modle 目录下。

6.9.3. Lib_* 框架附加类自动加载

框架附加类不是框架的必须功能，随框架一起发布，在用户需要的情况下手动调用并自动加载的类。该类将会自动寻址到/_frm/library 目录.

6.9.4. 其他自动加载类目录

项目的全局用户自定义类和第三方类库，以及与子系统相关的用户自定义类和第三方类库目录，即：

- /_inc/class
- /_inc/library
- /system/*/_inc/class
- /system/*/_inc/library

这些目录下的类定义文件将会在需要的时候(初始化类对象时)被自动加载。用户也可以通过框架的 Core::addClassSearchPath(目录绝对路径);

方法添加自动加载目录.

6.9.5. 自动加载类的层级寻址

在自定义的类中，下划线'_'带有特殊的含义，表示寻址中的层级结构，例如：

Model_User_FriendShip 将会寻址到/_inc/model/User/FriendShip.class.php，

Module_Service_User_Register 将会寻址到/_inc/module/Service/User/Register.class.php，

最后自定义类 Thrift_Membership 将会在在以下文件中依次寻找(假设当前子系统为 admin)：

- /_inc/class/Thrift/Membership.class.php
- /_inc/library/Thrift/Membership.class.php
- /system/*/_inc/class/Thrift/Membership.class.php
- /system/*/_inc/library/Thrift/Membership.class.php

需要特别提醒的，控制器也支持层级关系，但是控制器是以点号'.'分隔目录与文件名，且由于控制器名默认通过用户提交的__c 参数(也可以在入口处进行自定义参数名)来识别，因此建议统一使用小写形式。例如：

user.login 将会寻址到控制器目录下的/user/Login.class.php 文件，

usercenter.friends 将会寻址到控制器目录下的/usercenter/Friends.class.php 文件，

以此类推。

6.10. 类单例模式

继承于 Base 的类具有单例模式的功能，但需要在类中定义一个公共方法用户获取单例对象：

```
/**
 * 获得单例.
 *
 * @return 类名称
 */
public static function instance()
{
    return self::InstanceInternal(__CLASS__);
}
```

这样类在使用的时候可以通过 **类名::instance()** 获得单例实例，注意注释中的**@return** 需要写明类名，这样在 IDE 的代码提示中才能够得到有用的提示信息。

6.11. 日志工具类

日志工具类由/_frm/core/component/Logger.class.php 文件提供，具体请查看该文件。另外在 Base.class.php 中也有对日志的封装，如果需要实现不同的日志存储逻辑，可以在派生类中覆盖其 log 方法。

6.12. 调试工具类

调试工具类由/_frm/core/component/Debugger.class.php 文件提供，具体请查看该文件。