



# Fast,Compatible & Simple OO PHP Framework Developer Manual

## 面向对象的 PHP 开发框架 FCS 开发人员指南



版 本 号	0.7.0
提交单位	世纪流年工作室
作 者	刘晨 <流年>
联系方式	Liu21st [at] gmail.com
完成日期	2006 年 03 月 15 日
文档版本	Rev.1.0

.....



# 第1部分 版权信息

发布本资料须遵守开放出版许可协议 1.0 或者更新版本。

未经版权所有者明确授权，禁止发行本文档及其被实质上修改的版本。

未经版权所有者事先授权，禁止将此作品及其衍生作品以标准（纸质）书籍形式发行。

如果有兴趣再发行或再版本手册的全部或部分内容，不论修改过与否，或者有任何问题，请联系版权所有者 liu21st[at]gmail.com。

对 FCS 有任何疑问或者建议，请进入 FCS 讨论组 [ [http://groups.google.com/group/FCS\\_PHP/](http://groups.google.com/group/FCS_PHP/) ] 发布相关讨论。并在此感谢 FCS 讨论组的所有成员。

有关 FCS 项目及本文档的最新资料，请及时访问作者网站

世纪流年 <http://blog.liu21st.com>

或者 FCS 项目主站 <http://labs.liu21st.com/index.php/m,fcs> 。

文档规范约定：

请注意文档中相关信息提示以及各自代表的含义



表示提示信息



表示需要注意的事项



词语解释



给出的友好提示



可能存在的 BUG



需要引起注意，否则会导致问题



错误的操作

## 目 录:

第 1 部分	版权信息.....	2
第 2 部分	框架概述.....	5
2.1	FCS 概述.....	5
2.2	系统特色.....	5
2.3	目录结构.....	15
2.4	系统架构.....	17
第 3 部分	开发指南.....	19
3.1	系统配置.....	19
3.2	数据模型.....	23
3.3	数据访问.....	25
3.4	模板输出.....	27
3.5	添加语言包.....	29
3.6	错误调试.....	30
3.7	应用开发.....	30
3.8	应用部署.....	37
3.9	系统基类库.....	38
第 4 部分	模板指南.....	44
4.1	模板概述.....	44
4.2	模板结构.....	44
4.3	模板定义.....	45
4.4	模板标签.....	46
4.5	模板变量.....	46
4.6	模板注释.....	48
4.7	公共模板.....	48
4.8	对象标签.....	49
4.9	列表标签.....	49
第 5 部分	附录.....	51

5.1	文件列表.....	51
5.2	类库列表.....	52
5.3	函数列表.....	53

## 第2部分 框架概述

### 2.1 FCS 概述



FCS 全称 **Fast , Compatible & Simple OOP PHP Framework**

作者注：快速、兼容而且简单的面向对象的轻量级 PHP 开发框架

通过 FCS 框架您可以轻松地构建中小规模的基于 WEB 的 PHP 应用。

发布此框架的目的是让 **PHP** 的 **WEB** 应用开发更加简洁和快速，并希望能够不断吸收和融入更好的技术！不采用庞大和过于抽象的类库，全部自主化编程，简洁、快速和实用是 **FCS** 发展秉承的宗旨！

系统的主要特点可以从下面三大方面来体现：

- 一、快速：快速的应用开发和优化的执行速度，更加有利于企业级应用和部署
- 二、兼容：兼容 PHP4 和 PHP5 版本，在 PHP4 下模拟了很多 PHP5 的优秀特性
- 三、简单：简洁的架构和模板制作配合全中文注释，让你的应用变得更加简单



注意：本文档针对的 FCS 版本为 **0.7.0**，该版本在之前版本的基础之上作了较大的调整，包括架构思想和实现方法，使用过之前版本的朋友应重新学习。

### 2.2 系统特色

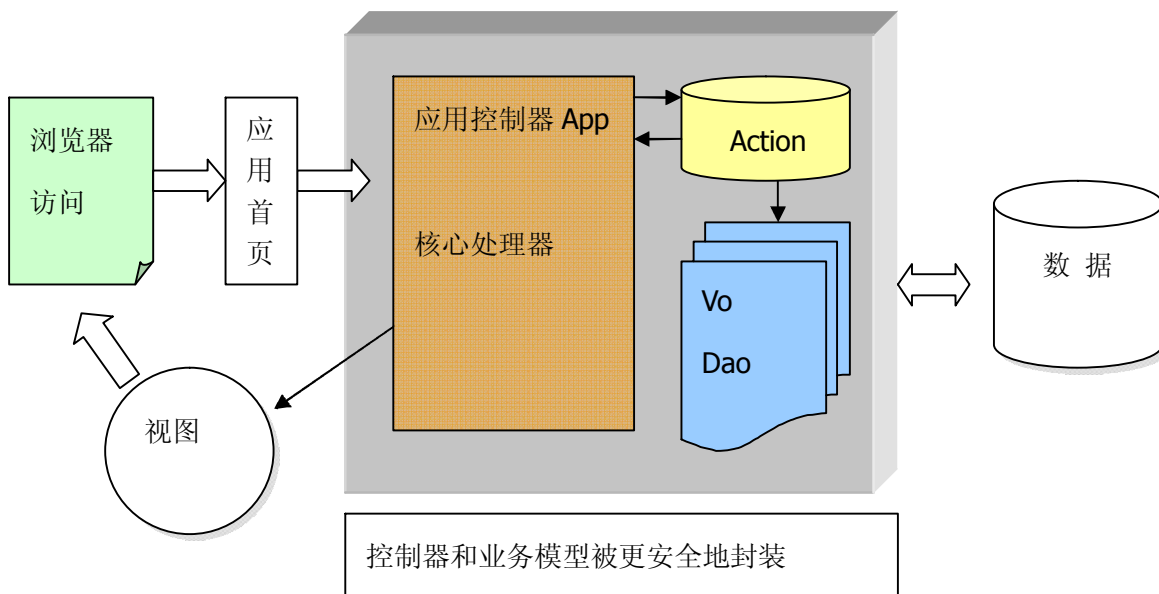
FCS 框架包含很多特性，包括对 PATHINFO 的全面支持、命名空间方式导入类库、编码转换和 UTF-8 支持、国际化支持、动态缓存和静态生成、日志和错误调试、多项目的支持以及对企业级开发和部署的支持。并且在可维护性、安全性、可移植性以及可扩展性方面都有很好的表现。

### 2.2.1 面向对象和 MVC 模式

面向对象是 FCS 框架的实现基调，除了个别必须的公共方法之外，框架的全部文件都采用类的形式实现，为以后的维护和扩展带来了方便。整个应用系统采用单一入口方式访问，其它组件和类库被有效的封装到一个安全的地方，无法通过 WEB 浏览器访问到，只有入口程序和模板文件在 WEB 目录下面。因为采用了面向对象的开发方式，FCS 系统的可维护性、可扩展性和可移植性方面表现都非常出色。

⚠ 注意：考虑到兼容的需要，FCS 的面向对象不是纯 OO 的实现，为了在 PHP4 环境下的面向对象的程度更加加强，作了很多的完善和模拟，并且在不牺牲性能的前提下，实现了跨版本的面向对象实现和良好运作。

和很多应用框架一样，FCS 也采用了 MVC 模式，但是在实际过程中，做了些调整，控制器由应用控制器和 Action 控制器类（和 App 并无继承关系）构成，Dao 和 Vo 类库实现了模型，模板则实现了视图的概念。并且系统的控制部分和模型部分被更安全地封装起来，放到 WEB 无法访问的路径下面。下图给出了大致的框架模型。



模型的实现可以比较灵活，如果是一个相对较小的应用，甚至可以没有必要通过 Vo 对象和 Dao 类来实现，你可以通过访问抽象数据类 Db 来直接返回一个数据集，这种方式对于比较小的应用反而

更加直接。但是对于企业级的应用开发，就必须依赖多个 Vo 对象和 Dao 对象才能很好的控制和规范项目，当然，这也是系统推荐的方式，无论是简单或者复杂的应用都可以胜任。一般情况下，根据业务情况添加不同的 Vo 对象和 Dao 对象就完成了模型的工作，大部分的业务模型都由 Dao 来完成，然后由 Action 来封装和调用。

应用控制器（核心控制器）和 Action 控制器都承担了控制器的角色，开发人员主要涉及到 Action 控制器的开发，而且相当简单，只是添加不同 Action 的业务逻辑控制，调用由应用控制器来负责。

系统的视图（模板）的实现是和框架无关的，做到了 100% 分离，可以独立预览和制作。内置的模板引擎给模板的制作带来了更方便有效的方法。

## 2.2.2 命名空间和国际化支持

FCS 的所有类库引用采用命名空间的方式，以保证类库的唯一性。在应用开发的时候不需要使用传统的 includeh 或者 require 指令，取而代之的是使用 Import 方法（为了考虑 .Net 的开发人员，另外 Import 方法还有个别名 using），Import 方法导入的根目录是系统的类库目录 LIB\_DIR。只要遵循命名规范，类库的导入和使用就显得方便有效，如

```
Import("FCS.util.ListIterator");
```

```
Import("com.liu21st.index.Vo.UserVo");
```

Import 方法支持多个文件引入和子目录引入

```
Import("com.liu21st.index.Vo.*");
```

```
Import("com.liu21st.index.*");
```

但是系统不推荐使用子目录引入的方式，因为会给系统带来较大的目录遍历开销。



注意：在 Unix 或者 Linux 主机下面是区别大小写的，所以在使用 import 方法或者 using 方法的时候要注意目录名和类库名称的大小写，否则会引入文件失败。

由于命名空间设计的局限性，在命名目录名称的时候不能使用 "." 符号，否则会转换成 "/" 符号导致文件引入失败。

系统对国际化的支持比较好，支持语言包功能，您可以定制你需要的语言提示信息。

系统在启动应用的时候会检查当前用户的浏览器接受的语言，并会尝试在语言文件目录下面寻找该语言包定义文件，如果不存在则会采用默认的语言文件。

在检查到正确的语言后，系统会调用 setlocale 方法设置本地化支持（包括对时间、数字和货币的本

地化支持)。

系统自带了简体中文语言包文件，语言文件采用常量定义，您可以方便地扩展其它语言包文件。

Lang 目录下面的语言文件属于系统语言包，如果您的项目也需要定义语言包，请在 Lang 目录下面建立相应的项目名称 (APP\_NAME) 目录，并放置相应的语言包文件。

### 2.2.3 兼容 PHP4 和 PHP5

这是 FCS 框架最主要的特色之一，FCS 框架的实现是和环境与数据库无关的。系统在实现的过程中充分考虑了 PHP4 的环境，并且在不牺牲性能的前提下模拟的部分 PHP5 的实现方法，最大程度地保证了系统在运行过程中在 PHP4 下表现出的出色性能和在 PHP5 下优势的充分发挥。

由于先天的缺陷，PHP4 的面向对象机制不够完善，很多只有 PHP5 才能完成的面向对象特色如 Interface、访问控制、Abstract、Final 等在 PHP4 下无法运行，但是在有可能的情况下在其它方面作出了一定程度的 PHP5 特性模拟。

首先在 FCS 框架的基类 Base 类中实现了只有 PHP5 才支持的架构函数 \_\_construct () 和析构函数 \_\_destruct ()，并且对 PHP5 作了更大程度的支持，在 PHP5 环境下进行应用开发显得更加方便和便捷，例如魔术函数的实现增强了应用开发的容错性和自我完善性。非常明显的例子就是在 PHP5 环境下面，所有 FCS 基类库的引用都不必手动加载，由 \_\_autoload () 方法自动加载。另外一个情况就是 \_\_set () 和 \_\_get () 方法的实现，可以避免在定义数据对象的时候书写枯燥的 get 和 set 方法（因为数据对象都是 Public 的，并且 \_\_set () 和 \_\_get () 方法首先判断对象的属性是否存在）。

其次，系统模拟了一些 PHP4 中没有的而又是非常需要的方法和类库，包括 property\_exists ()、array\_combine () 等方法和 ListIterator 和 ArrayList、HashMap 等类库，某些类库是 SPL 中才有实现。

但是，兼容性的考虑在加强应用系统可移植性能的同时也不得不牺牲部分 PHP5 优秀的特性，这不得不说是 FCS 系统的一个缺憾。其一，是面向对象的机制得不到充分的发挥；其二，就是系统的异常捕获能力。但是，必须注意的是这些缺憾只是影响 FCS 基类库，而不会影响到应用类库，如果决定开发某个应用的时候，大致的部署环境已经确定了，那么在进行应用类库开发的时候就完全可以按照 PHP5 的特性来开发，上面所说的两个缺陷也在一定程度上弥补了，但是可移植性就降低了。

框架的基类库都基于 Base 扩展，这样充分保证了系统的兼容性。



注意：FCS 框架在 PHP 的 CGI/FastCGI 模式下需要更改设置，主要涉及到 \$\_SERVER["SCRIPT\_NAME"] 在该模式下面的值一般是 PHP 的执行程序。



## 2.2.4 项目部署和协作开发

FCS 框架为企业级的部署和开发提供了方便。

首先，FCS 框架无需安装，仅仅需要上传就可以了，而且支持 PHP4 和 PHP5，为了做到框架的通用性，在系统的核心设计上基本上没有使用特殊的函数和类库（因为某些特殊需要个别地方对 PHP4 不支持的函数作了模拟实现），并作了一定程度的 PHP5 模拟，让框架在 PHP4 下没有牺牲太多的性能，而且运行顺畅。

FCS 系统目录可以传到服务器的任何位置，例如 C:/FCS 或者 /User/Local/FCS

WebApps 目录传到你需要访问的 WEB 目录下面，由于 FCS 系统的架构特点，如果你有多个网站应用系统基于 FCS 框架构建，那么你不需设置多个网站目录，因为每个网站应用都采用单一入口（也就是说一个网站就只有一个入口文件，模板目录也是公用的，而且并不会相互影响），当然，由于其它原因（如由于不同域名的关系或者要放到不同的服务器）您仍然可以分开多个目录存放，这个并没有影响。



唯一可能需要设置的是静态文件目录每个网站应用必须分开设置。

如果您的服务器环境是 Unix 或者 Linux 系统，请保证下面的目录设置为可写权限 777，否则可能会出现异常：WebApps 目录下面的 HTML 目录、FCS 目录下面的 Cache 目录和 Logs 目录。

如果您没有服务器的管理权限，而只有普通上传权限，您也可以直接把 WebApps 目录和 FCS 目录放到相同的目录下面。

其次，系统无需任何配置修改就可以运行，让你的应用部署更加简单，当然，作适当的满足你需要的调整，可以让应用变得更实际。框架是简单的，但是可以构建复杂的应用。



注意：如果使用 Unix 或者 Linux 环境部署应用，需要对下面目录属性设置为可写，否则会造成异常。他们是：WebApps 目录下面的 HTML 目录、FCS 目录下面的 Cache 目录和 Logs 目录

新版的 FCS 框架把系统目录和网站目录分离，完全可以做到更安全的部署应用，您可以把 FCS 系统

目录放到非 WEB 访问目录下面，以保障应用的代码安全。新版类库的设计更加满足企业级应用开发中的协作和分布式开发的需要。每个项目、每个模块都独立目录，可以达到方便地集成。网站目录下面只有应用的首页文件，而且其中并没有任何业务逻辑处理，纯粹是系统应用的调用和初始化。

#### 应用部署和协作开发

不同应用项目的应用类库放在 FCS 系统目录的 Lib 目录下面

Lib 目录下面的 FCS 目录为基类库，可供各个项目调用。每个应用项目可以在 Lib 目录建立相应的命名空间，如：`com.liu21st.App1`、`Com.liu21st.App2`（实际的文件存放可能就是在 Lib 目录下面的 `com/liu21st/App1` 和 `com/liu21st/App2`。）各个项目的类库之间也可以相互调用，如：

```
Import ("com.liu21st.App1.Dao.UserDao")
```

```
Import ("com.liu21st.App2.Util.common")
```

这样做的目的是为了防止项目之间的冲突，因为可能公司的开发部门每年要给不同的公司做多个类似的项目，但是又存在区别，区别他们的方法就是使用唯一的命名空间来访问项目，当然你或许想用客户编号来区分不同客户的项目，那么也没有问题，你可以使用 `A00002.App1`、`B03458.App2` 的方式。除了可能存在的调用方式上的区别外，这样做没有其它区别。很显然第一种方式对于开发人员而言比较容易记忆。

FCS 支持分布式的开发和团队合作开发，体现在可以实现项目分布开发和模块分布开发，对于调试工作没有影响，最后项目整体完成后再统一集成。每个项目包是一个独立的目录，项目包下面的每个模块又相互独立，因此，协作和集成都很容易完成。每台开发服务器上面只要有 FCS 框架本身的基类库和框架文件就可以进行应用开发和测试工作。

基于 FCS 框架进行应用开发，开发人员只需要关注应用类库，所有的后台编码全部都集中（并且大部分情况下只有）在应用类库上面，在进行测试的时候需要模板文件的配合，模板文件的制作参考后面的模板指南。开发过程中涉及到的应用类库有几种区别，详细参考下面的开发指南。

同时，FCS 框架还为同时并发多个项目提供了良好的支持，主要表现在：

- ✧ 每个项目分配不同的入口文件，容易部署；
- ✧ 每个项目有单独的配置文件，相互独立，互不影响；
- ✧ 每个项目可以单独定制各自的语言包，不会冲突；
- ✧ 每个项目有独立的类库目录，分工协作更加方便；
- ✧ 每个项目的系统日志有单独目录，互不影响；

## 2.2.5 编码转换和 utf-8 支持

框架的文件全部采用 UTF-8 编码格式(为了更好的支持扩展,您以后的应用组件也应该遵循该规范),您可以配置输出的页面编码格式(默认采用 UTF-8 输出)

TEMPLATE\_CHARSET 模板字符集设置,默认使用 UTF-8 编码,如果你的应用模板文件使用的是 GB2312 或者其它编码格式,请设置相应的编码格式。

OUTPUT\_CHARSET 页面输出字符集 (默认采用 UTF-8 输出),如果你的应用要求采用 GB2312 编码输出,请更改相应的配置项。

系统内置了一个 autoCharSet () 方法来对编码进行自动处理,并且使用了 iconv 和 mb\_convert\_encoding 两种方式支持,为了提高效率,如果系统的 TEMPLATE\_CHARSET 设置和 OUTPUT\_CHARSET 设置项相同,则不会进行编码转换,因此,在不支持 mbstring 和 iconv 模块的服务器下,可以设置 TEMPLATE\_CHARSET 和 OUTPUT\_CHARSET 相同即可。

## 2.2.6 充分支持 PATHINFO

对搜索引擎友好的充分支持,框架会把所有的 GET 方式传值转换成 PATHINFO 形式,并且会把 PATHINFO 参数转换成 GET 数组变量,让您保持以前的传值和链接形式,转换的工作由框架来完成。您还可以关闭 PATHINFO 形式,甚至框架还支持你对 PATHINFO 地址的自定义格式,让你的 URL 地址富有创造性;

系统支持两种方式的 PATHINFO 地址,您可以根据应用的要求灵活地选用何种方式。

第一种: 参数对并联 /var1,val1/var2,val2/var3,val3/

第二种: 参数对串联 /var1=val1,var2=val2,var3,val3/

系统默认采用第一种方式,两种方式您都可以自定义参数和值之间的分割符号,只需设置 PATH\_DEPR 的值。

例如: 把 PATH\_DEPR 的值设置为 : 那么您的 URL 看起来会象下面的效果

http://<serverIp>/var1:val1/var2:val2/var3:val3/



注意: 请选择合理的 PATH\_DEPR 分割符号,否则会导致功能异常。

系统对 PATHINFO 访问技术做了很多安全性处理工作，例如访问某个模块的操作时候，需要跳转到其它模块，只需要改变模块参数，操作参数会自动去除，防止出现其它模块没有该操作出错的情况。当然，您确保始终保持传模块参数和操作参数是更加安全的方式。

PATHINFO 技术对以往的编程方式没有影响，因为 GET 和 POST 方式传值依然有效，不同的是在对 URL 路径的处理方式上，因为模拟了目录，所以在模板页面中对当前目录下面的连接处理不当可能会导致失效。如果你依然不打算采用 PATHINFO 方式，那么可以在项目配置文件中把 PATHINFO\_URL 的值设置为 False 就可以屏蔽掉了。

### 2.2.7 模板动态缓存

FCS 内置了一个性能卓越的模板引擎（由 Template 类实现），使用了动态编译和缓存技术，利用该模板引擎可以方便地定义模板文件，从而达到快速开发的目的。FCS 架构的设计中模板和程序完全分离，一套模板文件就是一个目录，模板是标准 html 文件（可以配置成其它后缀，如.shtml, .xml 等），可以单独预览，前提是所有外部文件引用和图片采用相对路径。

系统会在需要显示的过程中动态加载模板文件，并编译该模板文件，生成一个模板缓存文件，下次会自动检测该模板是否变更过，如果在缓存有效期之内并且模板没有改动过的话，系统就不会再次重新编译模板，而是直接读取编译过的缓存模板文件，编译过的缓存模板文件一旦包含之后就可以直接显示变量的值。所以，缓存模板中的显示内容依然是动态的（除了个别系统特殊变量的输出采用静态输出，因为其值比较稳定），而并不是一个静态的输出，如果您的应用需要采用静态文件输出，请设置启用静态文件功能，系统会定向到静态文件页面，当然，这样做法不适合数据更新比较频繁的应用。

由于使用了模板动态缓存技术，在您第一次运行某个组件的某个操作时候，对应的模板文件就会被缓存，下次读取的时候，无论是模板文件修改或者是缓存文件被删除，系统都会重新生成缓存文件。您可以设置模板缓存的有效时间间隔，如每隔 10 分钟重新读取模板文件。模板动态缓存只是让您免去每次重复编译模板的时间，数据本身并没有被动态缓存，因为数据的动态缓存涉及到应用的处理逻辑，所以并不打算在框架中集成，以后会考虑采用工具类库来支持。

FCS 内置模板引擎的模板标签相当简单，用过 Smarty 的模板标签的话，对该模板标签很容易掌握，Smarty 的标签简直是太复杂了，这给模板的制作带来了复杂性，FCS 的模板标签借鉴了部分 Smarty 的表现方式，而且在个别地方有所增强，模板文件主要应该由设计人员或者网页制作人员完成，因此过


于复杂的模板标签（尤其是逻辑标签）根本起不到时候。

使用{ \$var }的方式来定义变量，用{ include:filename }来包含外部文件，值得一提的是可以对模板变量设置函数功能（支持多个函数同时使用），以满足同一个变量不同方式输出的需要，而且变量没有赋值给模板引擎是不可能被输出的，有效地防止了数据的安全。还可以使用定义特殊变量，甚至可以输出PHP中定义的常量，另外模板标签的起始标记可以自己配置。

使用内置模板引擎的另外一个好处就是可以直接用标签来输出一个 Vo 对象甚至 VoList 对象，更加详细的模板说明请参考模板指南部分。

## 2.2.8 静态文件生成

静态页面生成和定向，如果您的应用数据对及时性要求不高，可以开启静态页面生成功能，系统会在第一次运行的时候生成静态页面，并且您可以对静态页面设置有效期，在有效期内你执行组件的操作时，页面会被重定向到静态文件，因为采用重定向技术，所以你无需对静态文件做目录更新操作，也无需担心静态文件被删除；系统在检测到模板文件被修改或者静态文件被删除的话，会自动重新编译生成静态文件。

 注意：如果采用静态文件重定向技术，你必须注意页面中的链接尽量保持绝对 URL 访问，如要跳转到某个地址采用 `http://labs.liu21st.com/?m=fcs&a=update`，而不要采用类似于 `?m=fcs&a=edit`（在非静态方式下是有效的）的方式，否则可能会造成链接无效的情况，因为当前处于静态文件 URL 而非 PHP 程序文件。

为了保证静态文件的唯一性，静态文件的生成也是按照模块来分目录存放，并对相同操作的不同参数页面生成多个静态文件，其规则是根据 GET 参数来生成。

## 2.2.9 抽象数据库访问层

FCS 自带的抽象数据库访问层支持多种数据库，并且可以很方便地扩展。目前系统支持的数据库包含有 MySQL、MySQLi 和 PgSQL，打算提供支持的还有 ODBC、Oracle 等，您可以根据我们的驱动规范来开发自己的数据库驱动类库。

通过数据库公共类库 Db 来访问底层数据库操作，或者可以通过更加抽象的 Dao 数据访问对象来操作数据库。

## 2.2.10 异常和错误处理

FCS 对异常处理的支持是完全的，无论是在 PHP4 或者 PHP5 环境下，您都可以使用 `ThrowException` 方法来抛出异常（考虑到简单性，系统并没有支持多种类型的异常捕获，这些如果需要的话，您可以在应用中单独扩展各个模块自身的异常处理方法），框架集成了 `AppException` 方法和 `AppError` 方法来处理



异常和错误，并且您仍然可以在这两个方法上作扩展处理，如果设置为非调试模式的话，抛出异常和错误的结果是定向到一个指定的错误页面，对于用户来说更友好些。

鉴于 PHP4 对于异常捕获的能力不够，所以在 PHP4 下面的异常捕获实现采用 `halt` 错误信息的方式来模拟，但是对于最终的效果是一样的，系统抛出异常后的页面显示信息 PHP5 和 PHP4 下效果一致。只是因为 PHP4 下面无法使用 `Try` 和 `Catch` 来自动捕获异常，需要手动判断抛出异常类。

系统的 `FCSException` 类是所有异常类的基础类，其它模块的异常处理类应该在该基础类上扩展。该类的实现进行了版本判断，如果在 PHP5 下面则 `FCSException` 类本身也是继承 `Exception` 类，否则就实现了一个模拟的异常类，这个要归功于 PHP 的 `debug_backtrace` 方法，是它让异常处理的 PHP4 实现成为可能。

默认 `ThrowException` 方法只能抛出系统异常，如果您要使用 `ThrowException` 方法来抛出自己扩展的异常处理，请添加 `$type` 参数来指定需要抛出的异常类型，如：`throwException('用户信息错误','UserException')`。

### 2.2.11 系统日志和调试

实现了简单的日志记录类，通过 `Log::Write($errorStr)` 方法来记录系统日志，包括系统异常和错误信息，您可以随时查看日志文件。日志文件分为 `SystemErr.log` 和 `SystemOut.log`。前者主要用于记录系统异常，通常为抛出异常或者捕获严重错误后自动记录，后者主要用于调试信息和页面的一些非严重错误记录，调试信息一般为 `systemOut` 方法写入。

在系统的调试模式中，系统的所有异常和错误都会记录到系统日志中，在应用实施后，您可以关闭调试模式，这样系统就不会自动完成日志记录，除非你自己触发日志写入。

系统对每个项目单独记录日志，所以查看的时候请注意定位到某个项目目录下。

如果您的应用组件需要记录特殊的日志，也可以调用（或者扩展）该方法来完成。

## 2.3 目录结构

0.7.0 版本开始的 FCS 目录结构和之前有所改变，关键是体现在企业级项目部署的方便性上。从 0.7.0 版本开始，FCS 框架除了模板目录、静态文件目录和网站入口文件必须放到 `WEB` 目录下之外，其它所有

框架的文件和目录可以单独存放，不受限制，您需要做的仅仅是在首页文件中指定 FCS 框架的包含目录，我们建议您如果可能的话把 FCS 框架的目录包放到其它网站不能访问的目录下面，以保障应用的安全性。

下面是 FCS 框架包含的目录结构：

├── WebApps 网站应用目录

├── index.php 首页文件

├── Tpl 模板目录

├── Html 静态文件目录

-----

├── **FCS** 系统框架目录（和网站目录可以分离）

├── Common 公共文件目录（存放一些公共函数）

├── Cache 系统缓存目录

├── Conf 配置文件目录（存放系统配置和项目配置文件）

├── Docs 文档目录

├── Lang 语言文件目录

├── Lib 类库文件目录（存放基类库和应用类库）

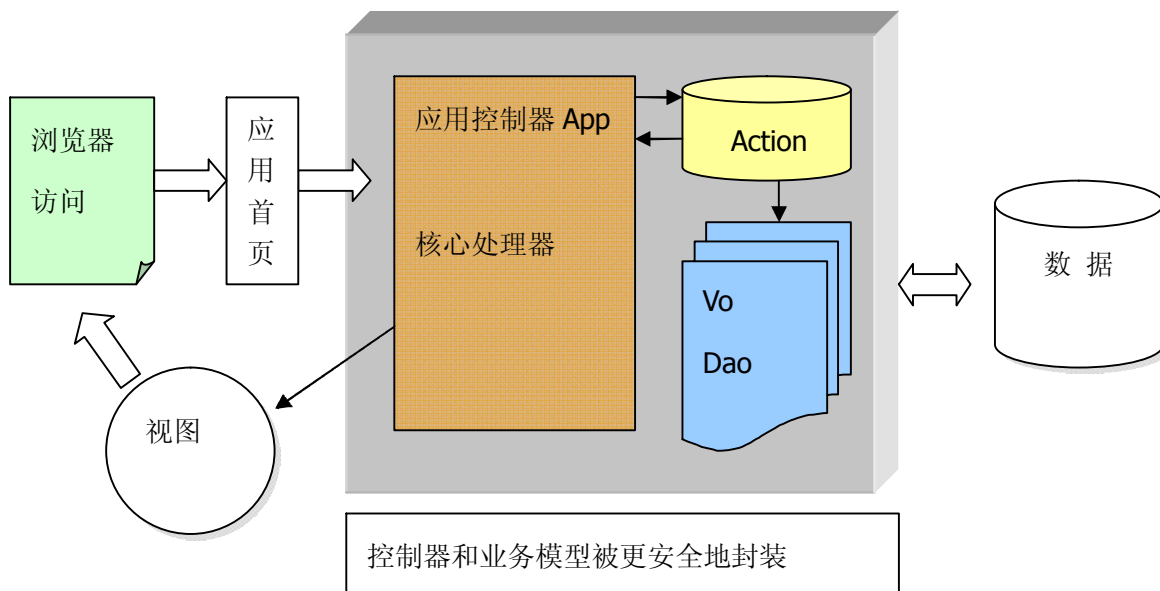
├── Logs 日志文件目录

├── PlugIn 插件文件目录



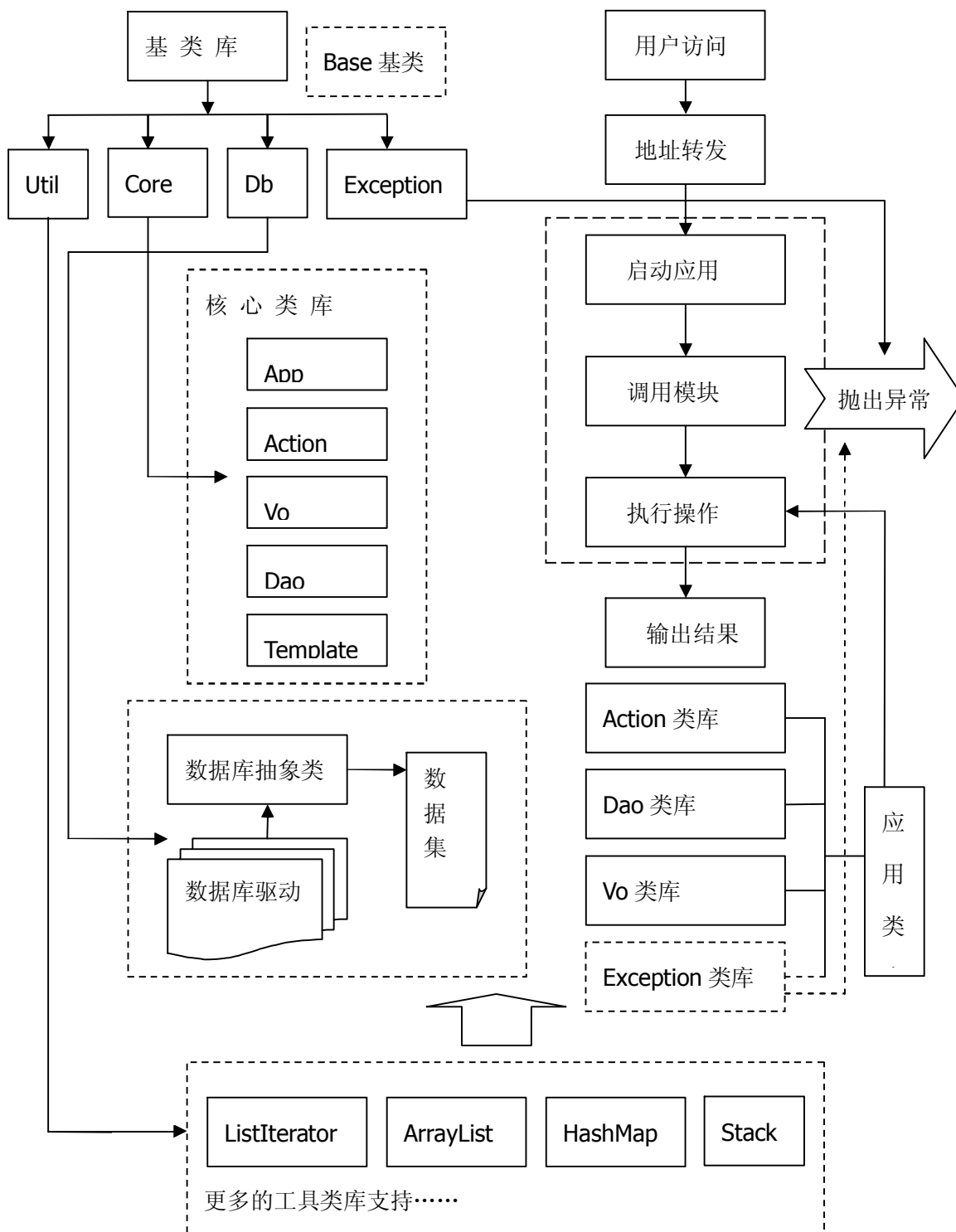
## 2.4 系统架构

FCS 系统把整个网站应用封装到一个 **WEB 容器** 中，该容器的访问入口是唯一的，找不到任何另外一个可以访问该应用的入口（在某种部署方式下即使是可以访问的，但是也不能开启应用），容器有一个后台端口，连接数据库进行数据交互，该容器本身由各类业务模型和视图、应用控制器组成。



并且 **FCS** 框架把控制器和业务模型移值到更加完全的地方（非 **WEB** 访问路径下）。系统在执行的过程中基于模块和操作来调用，每个 **Action** 目录下面的 **class** 都表示一个模块，其中的每个方法就表示一个 **Action** 操作。这样的模块概念并不大

FCS 系统的调用过程和类库架构如下：



## 第3部分 开发指南

本章描述了如何基于 FCS 框架进行快速应用开发，主要包括应用类库的开发和部署。

### 3.1 系统配置

在应用开发之前，需要做的工作还可能包括：网站首页文件、项目配置文件的配置。

一个标准的应用首页代码如下：

项目首页文件 index.php

```
// 定义 FCS 框架路径和网站路径 （使用绝对路径）
define('FCS_PATH', dirname(__FILE__).'./FCS');
define('WEB_ROOT', dirname(__FILE__));
define('APP_NAME', 'App1');
// 加载 FCS 框架公共文件
require(FCS_PATH."/FCS.php");
//实例化一个网站应用实例
$App = & new App();
$App->Init();
$App->Exec();
```

项目入口文件的名称并不一定要以 **index** 命名，只是为了便于访问，因此你一样可以用其它文件名，访问域名的时候添加项目入口文件名就可以了。

首页中的 **FCS\_PATH** 必须定义，否则无法定位 **FCS** 框架系统的路径，也就无从启动应用。如果把 **FCS** 框架的系统目录放到了其它非 **WEB** 访问目录下，需要在这里指定绝对路径。

**APP\_NAME** 就是定义了项目的名称，如果没有定义，系统默认取首页的文件名。

**WEB\_ROOT** 定义了网站的根目录，这里可以暂时不定义，在项目配置文件中定义。

根据项目的首页文件，你需要在配置目录下面增加和项目名称同名的项目配置文件，但是后缀不同，配置文件统一使用.ini.php。例如，上面的首页文件对应的配置文件就是 App1.ini.php。

框架本身自带了一个首页文件和配置文件的实现，可以提供参考。下面列出常用的一些系统配置项和项目配置文件的配置。

### 3.1.1 系统配置文件



一般情况下您不需要对系统配置文件进行修改，这些配置项的维护由 FCS 系统来完成，除非你需要更改 FCS 框架本身。

下面列出的项目供您需要的时候参考：

ID	配置项	说明
1	WEB_URL	网站访问目录，由系统自动获取，无需设置
2	CACHE_DIR	系统缓存目录名称
3	CONF_DIR	系统配置目录名称
4	LIB_DIR	系统类库目录名称，应用类库也包含在目录下
5	LOG_DIR	系统日志目录名称
6	FCS_VERSION	系统版本信息，系统升级会修改此项
7	APP_NAME	应用名称，系统自动获取

### 3.1.2 项目配置文件

除了系统配置文件之外，每个应用项目都有单独的配置文件，给应用提供更多的灵活性。

参考配置项如下：

ID	配置项	说明
1	WEB_TITLE	网站标题
2	WEB_DOMAIN	网站域名
3	WEB_ROOT	网站根目录位置


4	TMPL_DIR	网站模板目录名称
5	HTML_DIR	网站静态目录名称
6	DEBUG_MODE	是否为调试模式
7	ERROR_PAGE_URL	网站错误导向页面
8	DEFAULT_ERROR_MESSAGE	默认错误信息
9	PATHINFO_URL	是否使用 PATHINFO 的 URL 访问形式，系统推荐
10	URL_MODEL	URL PATHINFO 模式
11	PATH_DEPR	URL 参数和值之间的分割符号
12	VAR_MODULE	模块调用变量名称
13	VAR_ACTION	操作调用变量名称
14	DEFAULT_MODULE	默认调用模块名称
15	DEFAULT_ACTION	默认调用模块操作
16	DEFAULT_TEMPLATE_NAME	默认模板名称
17	TEMPLATE_CHARSET	模板字符集
18	OUTPUT_CHARSET	页面输出字符集
19	TEMPLATE_SUFFIX	模板文件后缀
20	CACHFILE_SUFFIX	缓存文件后缀
21	HTMLFILE_SUFFIX	静态文件后缀
22	COOKIE_DOMAIN	跨越有效的域名，用于会话保存
23	HTML_CACHE_ON	是否启用静态文件缓存
24	HTML_CACHE_TIME	静态文件缓存有效期
25	TMPL_CACHE_ON	是否启用模板缓存
26	TMPL_CACHE_TIME	模板文件缓存有效期
27	TMPL_L_DELIM	模板标签开始标记
28	TMPL_R_DELIM	模板标签结束标记
29	DB_TYPE	数据库类型
30	DB_HOST	数据库主机地址
31	DB_NAME	数据库名称

32	DB_USER	数据库用户名
33	DB_PWD	数据库密码
34	DB_PORT	数据库访问端口
35	DB_PREFIX	数据库表项目前缀

系统的配置项可能在以后的升级过程中会产生变化，这是正常的。

### 3.1.3 搭建项目环境

在完成应用首页和配置文件之后，就需要来搭建应用开发环境了，我们需要根据项目配置文件来构建应用类库的目录结构。首先，根据项目配置文件中的 **WEB\_DOMAIN** 配置项来建立相应的目录。

 注意：WEB\_DOMAIN 配置的域名是倒过来的，为了便于分析目录结构。如果你的域名是 [www.mydomain.com](http://www.mydomain.com)，那么这里的设置就应该是 **com.mydomain**。类似，如果域名是 [blog.liu21st.com](http://blog.liu21st.com)，设置就可以是 **com.liu21st**，二级域名 **blog** 应该作为你的项目名。

假如 **WEB\_DOMAIN** 的值是 **com.liu21st**，**APP\_NAME** 是 **App1**，那么我们就需要在系统类库 **Lib** 目录下面创建以下目录结构：

**com/liu21st/App1/Action/**

必须的，用来定位 **Action** 控制器，**Action** 目录名可在项目配置文件中设置为其它的。该目录下面部署了整个 **App1** 项目的 **Action** 控制器类，类名和文件名的命名规则是模块名+**Action**。例如，**UserAction**、**LeaderAction**，每个 **Action** 控制器中可以包含任意的 **Action** 操作方法，数量不限。每个 **Action** 控制器类并不一定和一个 **Vo** 对象或者 **Dao** 对象对应，只有当你的应用较大的时候，可以用来规范项目。对于小型的应用，你可以用一个 **Action** 控制器来控制多个 **Vo** 对象和 **Dao** 对象的操作（只不过在命名操作的时候需要有所规范）。

除了 **Action** 目录外，可能还需要增加下面目录（以下目录结构为可选或者以其它名称命名，因为所有文件都由 **Action** 控制器来加载，列出的为系统推荐目录）

**com/liu21st/App1/Dao/**                      放置 **Dao** 对象类

**com/liu21st/App1/Vo/**                      放置 **Vo** 对象类

com/liu21st/App1/Exception/ 放置异常处理类

你甚至可以增加其它目录如

com/liu21st/App1/Util/ 放置常用工具类

com/liu21st/App1/Common/ 放置公共类库等

不管建立何种目录，在 **Action** 控制器中的调用方式统一使用

```
Import('com.liu21st.App1.Dao.UserDao');
```

```
Import('com.liu21st.App1.Vo.UserVo');
```


等类似的方式，因此不同的目录只是在引用的时候不同。

应用开发目录就搭建好了，下面可以搭建模板文件目录

模板文件的目录在系统的模板目录 **Tpl** 下面创建下面目录结构：

Default/App1/Module

**Module** 的名称就是各个 **Action** 控制器的名称，每增加一个 **Action** 控制器类就需要添加一个 **Module** 目录。

 注意：缓存目录下的结构不需要手动创建，在运行的时候由系统自动创建。

如果项目需要独立的语言包，那么还需要在系统语言目录下面增加项目名称目录

如 Lang/App1/

## 3.2 数据模型

FCS 在开发过程中采用的数据模型是基于数据对象的。



数据对象：简称 **Vo**，是指对数据库中的某个数据表或者视图进行数据封装，以减少在数据访问过程中远程调用的次数，通过构建 **Vo** 对象，可以进行基于数据对象的数据访问。（此概念借鉴于 **Java** 中的 **Vo** 思想）

**Vo** 对象在 **PHP** 中的体现就是一个类，比如 **User** 表对应一个 **UserVo** 类，**Info** 视图对应一个 **InfoVo** 类等…，这个类的构造相当简洁，基本上就只是包含对应数据表的字段作为类的属性，其它的方法已经有基础类 **Vo** 完成，包括架构函数和 **ToMap** 函数等。

构造 **Vo** 对象的方法很简单，因为 **Vo** 对象的属性（必须）对应数据表或者视图中的字段，并且不需要包含其它任何属性，如果不需要对 **Vo** 对象进行特殊操作的话，可以不扩展任何的方法。**Vo** 基础类提供了两个方法：一个是架构方法，把某个数组、**HashMap** 或者其它对象转换成 **Vo** 对象；另外一个提供 **toMap** 方法，把 **Vo** 对象的属性转换成 **Map**，主要是供数据访问层操作。使得在进行数据对象操作的时候，可以非常方便地通过 **HashMap** 的 **get** 方法来获取某个字段的值，而进行条件匹配，实现复杂的查询功能。

在进行应用开发的时候，往往通过表单的提交值来组装成一个 **Vo** 对象，或者是要根据查询的结果来生成一个 **Vo** 对象，这些方法系统都内置实现了，都可以通过 **createVo** 方法来实现。当然，你还可以作一定程度的重载和扩展。

**CreateVo** 方法有三个参数：

<b>\$VoClassName</b>	需要创建或编辑的 <b>Vo</b> 对象的类名
<b>\$Type</b>	创建 <b>Vo</b> 对象的方式， <b>add</b> 新增 <b>edit</b> 编辑 默认为新增
<b>\$PK</b>	主键字段名，在编辑的时候用来取得 <b>Vo</b> 对象，默认为 <b>Id</b>

关于数据表的命名

在 **FCS** 框架的开发过程中，推荐使用下面的方式来命名数据表

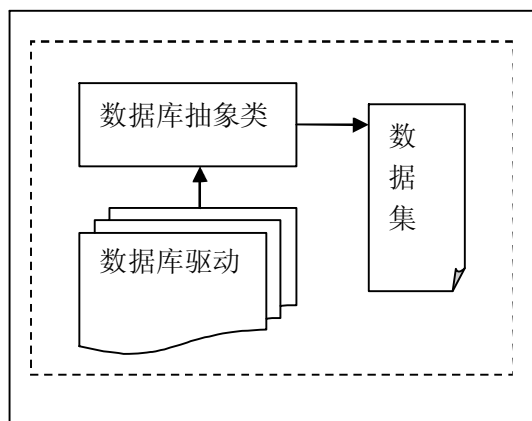
数据库前缀\_表前缀\_表名

可以映射为 **FCS** 框架中的：项目名\_模块名\_**Vo** 对象名，这三个概念在 **Dao** 基础类中有相应的属性。



### 3.3 数据访问

FCS 没有使用 Adodb 或者 PDO（PHP5 才内置支持）作为抽象数据库访问层，而是内置实现了一个抽象数据库访问层，当然这个抽象层的实现上也参考了部分目前比较先进的抽象数据库访问层框架，这个抽象层会处于不断完善的过程中。



抽象数据库访问层的关键类库在于数据库的公共访问类 **Db** 类，这个类是操作数据库的底层接口，换句话说，**Db** 类不能够独立存在，必须有对应的数据库驱动类库，但是开发人员不能够跨过该类直接进行数据库操作的访问。系统提供了目前几个主要的数据库的驱动类库，包括了对 **Mysql**、**Mysqli** 和 **Pgsql** 的支持，更多的数据库驱动也在完善中。开发人员也可以按照规范编写自己的数据库驱动（甚至包括众多的嵌入式数据库的驱动）。数据访问的结果保存在数据集对象中。

数据库的配置文件有多种定义方式

第一种 在每个应用的配置文件中定义

//数据库设置

```
define('DB_TYPE', 'mysql');
define('DB_HOST', 'localhost');
define('DB_NAME', 'web');
define('DB_USER', 'root');
define('DB_PWD', '');
define('DB_PORT', '');
define('DB_PREFIX', APP_NAME);
```

系统推荐使用该种方式，因为一般一个应用的数据库访问配置是相同的。

第二种 使用 **DSN** 方式在初始化 **Db** 类的时候传参数

```
$db_dsn = "mysql://username:passwd@localhost:3306/DbName";  
$db = new Db($db_dsn);
```

第三种 使用数组传参数

```
$DSN = array(  
    'dbms'      => 'mysql',  
    'username' => 'username',  
    'password' => 'password',  
    'hostname' => 'localhost',  
    'hostport' => '3306',  
    'database' => 'dbname'  
);  
$db = new Db($DSN);
```

在初始化数据库公共类的时候，会根据 **dbms** 的值来定位数据库类型，并进行数据库连接，如果数据库驱动不存在或者连接失败，会抛出异常。

在进行应用开发的实际过程中，如果使用数据模型 **Vo** 对象的话，并不需要直接操作该抽象访问层，而只需要通过各自的 **Dao** 对象来访问和操作数据库。**Dao** 对象属于抽象数据库访问层之上的业务封装，这样也更加完全和方便。

**Dao** 基础类实现了一些常用的操作，例如

1、**Add(\$Data,\$Table)** 新增一个数据对象

**\$Data** 可以是一个 **Vo** 数据对象，也可以是一个 **Map** 对象

**\$Table** 可以指定数据表的名称，默认会根据当前的 **Dao** 对象来分析取得表名

之前提到了数据表的命名方式建议，**Dao** 基础类定义了三个属性，其中 **tableName** 会根据当前对象自动获取，其它两个属性需要赋值，默认为空。**Dao** 类中的 **getFullTableName()** 方法就是取得数据库中的实际数据表名。

```
var $appPrefix;      //项目前缀  
var $modPrefix;      //模块前缀  
var $tableName;      //数据表名
```

## 2、Save(\$Data,\$Table) 保存一个数据对象

该方法的用法类似于 Add 方法，只不过用于 Update



注意：在新增操作的时候，给主键定义的值将无效（系统默认的主键是自动增长类型的，无需手动赋值），同样，在保存方法的时候，给主键定义的值将会转换为条件来定位到该数据记录，并且会删除该值的保存，这些都是系统自动完成的。

## 3、DeleteById(\$Id,\$Table) 根据主键的值来删除一个数据对象

\$Id 是数据对象的主键的值

Dao 类中有一个专门定义主键名称的属性\$PK，默认为 Id，如果需要可以设定为其它值

## 4、Delete(\$Condition,\$Table) 根据条件 Map 删除数据对象

\$Condition 是一个 HashMap 对象

## 5、getId(\$Id,\$Table,\$Fields) 根据主键得到 Vo 对象

## 6、find(\$Condition,\$Table,\$Fields) 根据条件查询取得 Vo 对象

## 7、findAll(\$Condition,\$Table,\$Order) 根据条件查询取得 VoList 对象

# 3.4 模板输出

处理完业务逻辑后需要将相关的变量输出到页面中，在调用模板显示之前，就需要将设置用于模板输出的变量，其它任何没有经过设置的变量是不会在模板页面中输出的（注：系统变量和特殊变量除外，见模板指南的模板变量部分）。基本上，模板变量的分配有三种类型，而且用法都类似。

第一种，普通变量的输出

```
$tpl = new Template();  
$value = '世纪流年';  
$tpl->assign('name',$value);
```

经过这样的变量分配之后，在模板文件中按照模板变量的定义规则就可以正常输出变量的值了。

在模板文件中可能的写法是

`{ $name }` 或者 添加函数输出 `{ $name|md5 }`

要注意的是分配的变量输出是使用 `echo` 的方式输出，如果存在非字符串方式的值，系统会禁止输出，以防止输出对象或者其它形式不想得到的输出。更多的模板变量信息参考模板指南部分。

第二种，Vo 对象属性的输出

由于系统的特性，使得要输出一个 Vo 对象的属性变得相对麻烦（如果你只是要输出 Vo 对象的某个属性还可以考虑），所以 FCS 系统专门为 Vo 对象的属性值的输出定制了一个 `assignVo` 方法，参考下面的例子。

```
$tpl = new Template();
$userVo = new UserVo($Map);
$tpl->assignVo('user',UserVo );
```

经过上面的赋值后，我们就可以在模板文件中定义下面的标签来输出 Vo 对象的属性了。

下面的标签格式都是有效的：

```
{Vo: user| username}
{Vo: user| $username }
{vo: user| username | substr(###,8) }
```

第三种，是 Vo 对象列表的输出，主要用于输出查询数据集

如果要输出一个数据集呢，如果使用对象标签岂不是累的要死，呵呵～所以，系统专门为数据集的列表输出定义了列表标签。其实，数据集就是包含多个 Vo 对象的 `ArrayList` 对象。


Vo 对象列表的输出方式

```
$tpl = new Template();
$dao = new UserDao();
$userList = $dao->findAll();
$tpl->assignVoList('user',$userList);
```

在模板文件中的标签定义方式参考下面

```
{voList : user | <div>$id $username $email </div> };
```

循环部分可以使用 HTML 标签,每个记录的值可以用变量的方式定义,但是不需要再添加 { 和 } 前后标记。

 注意: assignVo 和 assignVoList 方法只能用于输出 Vo 对象(及其派生类)和 VoList 对象(其实就是包含 Vo 对象的 ArrayList 对象),用其它变量进行赋值会导致异常。

## 3.5 添加语言包

如果不考虑多语言的支持,那么您可以直接把需要的信息文字写在程序中,如

```
ThrowException('新增用户失败!');
```


如果你的应用涉及到国际化的支持,那么可以定义相关的语言包文件。任何字符串形式的输出,都可以定义语言常量。框架的系统语言包目录在系统框架的 Lang 目录下面,每个语言都对应一个语言包文件,系统默认只有简体中文语言包文件 zh-cn.php,如果要增加繁体中文 zh-tw 或者英文 en,只要增加相应的文件。

语言包采用常量定义的方式包含,语言包的使用由系统自动判断当前用户的浏览器支持语言来定位,如果找不到相关的语言包文件,会使用默认的语言。如果浏览器支持多种语言,那么取第一种支持语言。一个定义语言包的信息项的方法是

```
define('ADD_USER_ERROR', '新增用户失败!');
```

这样一来,在程序中的写法就要改为

```
ThrowException(ADD_USER_ERROR);
```

 注意:为了不和其它项目冲突,每个项目可以单独定制自己的语言包,这些语言包文件,放到 Lang 下面的项目名目录下面。定义方式和系统语言包一样。

## 3.6 错误调试

在应用开发的过程中，如果需要做一些调试工作，可以输出一些调试信息到日志文件来模拟一定程度的调试（更加完善的调试请使用调试框架或者某些开发环境下的调试器）。

输出调试信息的方法是

在需要调试错误的地方使用 `systemOut` 方法，参数是一段字符串信息。该方法记录了调试的时间和输出的调试信息。

输出结果可以在框架的日志目录下面的 `systemOut.log` 文件中看到，如

06-03-10 17:17:59 [调试] SQL = INSERT INTO user(username) VALUES ('流年')



注意：如果启用数据库的调试模式，在运行的时候执行的每一个 SQL 指令都会记录到 `systemOut.log` 文件中。

## 3.7 应用开发

基于 FCS 框架的应用开发过程大致有下面几个过程：

1. 数据模型分析，构建 Vo 对象
2. 根据应用的程序，构建不同程度的 Dao 对象
3. 如果可能的话，创建每个模块的异常扩展（该过程可选）
4. 调用 Dao 对象，实现各个业务逻辑 Action
5. 如果业务逻辑特别复杂，或者 Dao 对象比较多，也可以考虑构建模块的 Helper 类，便于管理
6. 创建各个业务逻辑实现需要的页面模板

下面尝试给出一个基于 FCS 框架进行应用开发的范例。本范例是对一个用户表进行操作，多表实现的例子可以根据数据模型来扩展 Dao 和 Vo 对象实现，方式是类似的。

### 3.7.1 创建 Vo 对象

暂时没有关联，所处的功能模块为 INFO 模块，假设该项目的数据库前缀是 OA，那么我们定义各个

表的表名如下

用户表                      OA\_INFO\_USER

用户表 (OA_INFO_USER)
编号 (Id) pk
姓名 (username)
密码 (Password)

首先通过数据库工具（如 phpMyadmin）创建数据表

```
CREATE TABLE `oa_info_user` (  
  `Id` SMALLINT( 5 ) UNSIGNED NOT NULL ,  
  `username` VARCHAR( 20 ) NOT NULL ,  
  `password` VARCHAR( 32 ) NOT NULL ,  
  PRIMARY KEY ( `Id` ) ,  
  INDEX ( `username` ) ,  
  ) COMMENT = '用户表'
```

根据我们设计的数据模型，我们设计 UserVo 对象如下

```
class UserVo extends Vo{  
    // 要显示的字段  
    var $Id;  
    var $username;  
    var $password;  
}
```

因为不需要对某个字段作特殊处理，所以直接继承 Vo 基础类的相关方法就可以了，因此该 Vo 对象显得比较简单，主要就是定义需要在视图中显示的各个数据表字段。



注意：这个 **UserVo** 对象中属性和数据表中的字段名称是相同的，这并非必须，而是为了在初始化和转换成 **Map** 对象的时候方便，否则就会有额外的工作。如果你的数据表设计多个表之间存在相同的字段名，而且必须关联到同一个 **Vo** 对象中，可能就需要重载 **Vo** 基础类的某些方法了，包括 **\_\_construct** 和 **toMap** 方法（尤其是 **toMap** 方法，因为该方法用于数据库查询用的）

### 3.7.2 构建 Dao 对象

该应用属于单表操作，不需要构建 **Dao** 对象，或者我们可以构建一个空的 **UserDao**，以便于以后扩展更多的方法。

```
class UserDao extends Dao{  
    //不需要添加架构函数和任何方法  
}
```

### 3.7.3 添加异常处理

异常处理很简单，直接继承系统异常类就可以。

```
Classs UserException extends FCSEException{  
    //不需要添加任何方法  
}
```

### 3.7.4 实现操作方法

在完成 **Dao** 对象后，就可以添加需要的 **Action** 类了，一个 **Action** 类就可以表示一个模块（也就相当于是一个 **Action** 控制器）。下面分别对几个主要的操作讲述下如何定义 **Action** 控制器。我们新增一个 **Action** 控制器类，名称为 **UserAction.class.php**，该模块的访问方法就是：

`http://<serverIp>/?m=user` 访问 **user** 模块的默认方法（假设网站入口文件是 **index.php**）

`http://<serverIp>/?m=user&a=add` 访问 **user** 模块的 **Add** 方法



开始建立的内容为：

```
Import ('com.liu21st.App1.Dao.UserDao');  
Import ('com.liu21st.App1.Vo.UserVo');  
Class UserAction extends Action{  
  
}
```

下面要做的就是 在 **UserAction** 控制器中添加不同的操作方法。

新增用户的方法比较简单，主要是显示新增模板页面，没有其它变量需要赋值。

//新增用户

```
function add(){  
    $this->Display();  
    return;  
}
```

新增页面提交后就会转到 **insert** 方法，下面是实现范例。

//插入用户

```
function insert(){  
    $user = $this->createVo('UserVo');  
    $dao = new UserDao();  
    if($dao->add($user)){  
        $this->assign("message",'新增成功');  
        $this->assign("jumpUrl",'?a=index');  
    }else {  
        ThrowException ('新增用户失败！');  
    }  
    $this->forward();  
}
```

这里解析下 **forward** 方法的使用，**forward** 方法主要用于操作后提示信息和页面的跳转。本身也利

用了模板技术，在 `forward` 方法之前，可以对相关页面变量赋值，这些变量包括：

`Message` 要提示的文字信息 ，可以包含 `html` 标记

`jumpUrl` 要跳转的页面，如果没有定义，默认返回前一页

`waitSecond` 页面自动跳转时间（秒）如果没有定义，默认为 `3s`

`closeWin` 是否需要关闭窗口 如果设置为 `1`，则在提示信息后自动关闭窗口

新增用户成功后，就可以进行用户的编辑操作了。

//编辑用户，比新增用户多了取出要编辑用户的信息

```
function edit(){  
    $Condition = new HashMap();  
    $Condition->put("Id",$_GET['id']);  
    $dao = new UserDao();  
    $user = $dao->find($Condition);  
    $this->assignVo("user",$user);  
    $this->Display();  
    return;  
}  
//更新用户资料  
function update(){  
    $user = $this->createVo('UserVo','edit');  
    $dao = new UserDao();  
    if($dao->Save($user)){  
        $this->assign("message",'更新成功');  
        $this->assign("jumpUrl",'?a=index');  
        $this->forward();  
    }else {  
        ThrowException('编辑用户失败！');  
    }  
}
```

```
// 删除用户

function delete(){

    $Condition = new HashMap();

    $Condition->put("Id",$_GET['id']);

    $dao = new UserDao();

    if($dao->delete($Condition)){

        $this->assign("message",'删除成功');

        $this->assign("jumpUrl",'?a=index');

        $this->forward();

    }else {

        ThrowException('删除用户失败！');

    }

}

//首页 显示全部的用户信息列表

function Index()

{

    $dao = new UserDao();

    $userList = $dao->findAll();

    $this->assignVoList('user',$userList);

    $this->Display();

    return;

}
```

### 3.7.5 定制模板页面

下面假设模板页面使用 UTF-8 编码格式。

首页的模板文件内容：

```
{include:header}

<h3>用户列表</h3>
```

```
<div>
{voList:user |
<H4> $username <A HREF="?a=edit&id=$id"> 编辑</A> <A HREF="?a=delete&id=$id"> 删除
</A></H4>
}
</div>
{include:footer}
```

新增页面的模板文件内容:

```
{include:header}
<h3>新增用户</h3>
<div>
<H4>输入用户数据</H4>
<FORM METHOD=POST ACTION="">
姓名: <INPUT TYPE="text" NAME="username"><BR>
密码: <INPUT TYPE="text" NAME="password">
<INPUT TYPE="hidden" name="a" value="insert">
<INPUT TYPE="submit" onclick="add()" value="保存" >
</FORM>
</div>
{include:footer}
```

编辑页面的模板文件内容:

```
{include:header}
<h3>编辑用户</h3>
<div>
<H4>输入用户数据</H4>
<FORM METHOD=POST ACTION="">
<INPUT TYPE="hidden" NAME="id" value="{vo:user|id}">
```

```
姓名: <INPUT TYPE="text" NAME="username" value="{vo:user|username}"><BR>
密码: <INPUT TYPE="text" NAME="password" value="{vo:user|password}">
<INPUT TYPE="hidden" name="a" value="update">
<INPUT TYPE="submit" value="保存">
</FORM>
</div>
{include:footer}
```

Header 文件的内容

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>{$FCS.const.APP_NAME}</title>
</head>
<body>
```

Footer 文件的内容:

```
</body>
</html>
```

## 3.8 应用部署

应用开发完成之后的部署工作非常简单，基本上只需要上传然后适当做些配置修改就可以了。把整个 FCS 系统目录（包含了集成后的应用类库）拷贝或者上传到指定的目录，然后把项目首页文件和模板目录整个上传到 WEB 访问的指定目录下面（注意相关目录的权限），如果项目配置文件的某些设置（例如，文件路径和测试环境有所区别、数据库配置修改等）需要调整，再做一些调整就可以了。



注意：确保您的项目开发测试环境和应用部署环境一致，因为系统框架本身兼容没有问题，但是如果您的应用类库开发过程中没有考虑系统的兼容性问题，那么部署后的项目可能和测试环境有所区别。

建议的有些设置项最好在项目实际运作的时候修改，例如：

`DEBUG_MODE` 关闭调试模式 设置为 `False`

因为一方面，调试信息的输出对用户不够友好，另外一方面，频繁地写日志文件会增加系统的开销。

`TMPL_CACHE_ON` 启用模板缓存 设置为 `True`

因为在调试的时候为了调试方便，可以关闭模板缓存。实际运作的时候模板动态缓存可以减轻一部分系统负荷。

## 3.9 系统基类库

FCS 框架通过基类库的概念把所有系统类库都集成在一起，包括 FCS 的核心类库。基类库的作用是完成框架的通用性开发而必须的基础类和常用工具类等，包含有：

- ✧ FCS.core 核心类库包
- ✧ FCS.db 数据库类库包
- ✧ FCS.util 工具类库包
- ✧ FCS.exception 异常处理类库包

### 3.9.1 如何引入系统类库

FCS 框架中引用基类库和应用类库的方式采用命名空间引用方式（当然，如果你愿意一样可以通过 `include` 或者 `require` 方式包含类库文件，不过那样的话也许更加麻烦），系统集成了 `Import` 方法来引用类库文件（包括系统基类库和应用类库）。`Import` 方法默认的命名空间根目录是 `Lib`（系统配置中的 `LIB_DIR` 项）目录，假如要引入 `Lib/FCS/util/` 下面的 `HashMap` 类，可以通过使用

`Import ("FCS.util.HashMap")` 来方便地引入该类

或者使用 `Import ("FCS.util.*")` 来引入 `util` 目录下面的所有类库。并且支持多级目录引入，如

`Import ("FCS.*")`

但不推荐使用该方式，因为遍历多级目录的开销将会比较大。

整个 FCS 基类库会不断的完善和扩展，将来也许不止这些包，该类库的更新由 FCS 系统来完成，无需用户进行扩展。用户的应用级别的扩展可以放到项目的工具包中。当然，如果你有兴趣并且有足够的可以进行其它基类库的扩展，如

`IBM.core`

IBM.util 等类似的扩展

该基类库在项目应用中的访问方式是

```
Import ("IBM.util.*")
```

### 3.9.2 核心类库

核心类库是 FCS 框架运行的必须条件，该类库不可删除，其中还包括了重要的 Base 类。Base 类作为 FCS 系统的基类，在兼容性的实现方面作了很大工作。

- ✧ 模拟了 PHP5 才支持的类的\_\_construct () 和\_\_destruct () 方法
- ✧ 完成了自动变量设置和获取方法\_\_set () 和\_\_get () 方法
- ✧ 定义了输出对象属性数组的方法 \_\_varList ()
- ✧ 定义了输出对象属性值数组的方法 \_\_toArray ()
- ✧ 定义了输出对象初始化属性值数组的方法 \_\_toOraArray ()
- ✧ 定义了对对象的输出字符串方法 \_\_toString ()
- ✧ 定义了自动调用类的函数的方法 \_\_call ()

以上特性和方法可以被所有 Base 的子类继承和重载。



由于 Base 基类中实现了类的架构和析构方法，使得在 PHP4 中可以象 PHP5 一样使用\_\_construct () 和\_\_destruct () 方法，对 FCS 的兼容性有很大的功劳。


其次的核心类还包括：

App	[ 应用程序类 ]	应用程序控制器
Action	[ Action 基础类 ]	实现模块操作的一些基本方法和内置模板的接口
Template	[ 内置模板引擎类 ]	负责模板的解析和编译、缓存和输出
Vo	[ Vo 对象基础类 ]	实现数据对象和常用操作
Dao	[ Dao 对象基础类 ]	实现数据访问对象和基本操作

以上类库都派生自 Base 基类，所以说他们的地位是一样的，只不过分管不同的工作而已，在某个职责范围内看起来比较重要一点。

注意 Vo 和 Dao 的实现依赖于其它一些工具类库中的类，不过您在使用他们的时候不用再次包含他们了，因为在类的定义之前就已经 Import 了相关依赖包，在应用类库的开发过程中也应该养成首先

Import 依赖类库的习惯，虽然在 PHP5 下面可以通过\_\_autoload 方法自动加载类库。

 注意：系统定义的\_\_autoload 方法只是实现了 PHP5 下基类库的自动加载，应用类库的加载过程必须手动 Import 相关依赖包，这样做的目的是为了让开发人员清楚的知道 Action 需要哪些相关依赖包，便于调试。

### 3.9.3 数据库操作类库

数据库操作类库只有当您的应用使用了数据库作为存储对象的时候才需要（并非所有的应用都需要数据库的，呵呵～）。主要包含了数据库公共类、各类数据库驱动类和数据集对象。在实际应用开发过程中，你可能并不需要接触这些类库，只需要通过各自的 Dao 对象来访问和操作数据库就可以，Dao 对象处于抽象数据库访问层的上层，属于业务逻辑的实现层。Dao 对象的详细描述请参考数据访问章节。

数据库操作类库的主要特点有：

- 1、支持事务和回滚（当然首先需要数据库本身支持），但是默认是自动提交
- 2、在数据库调试模式下支持 SQL 语句的输出
- 3、完全支持中文 UTF-8 字符集
- 4、把查询操作和执行操作分开，便于返回不同的结果
- 5、支持配置文件和 DSN 两种数据库配置读取方式
- 6、支持对数据集的数组和对象两种方式返回
- 7、记录数据库查询次数和执行（写入）次数

数据库类库中最重要就是公共类 Db 的实现，除了引导各种数据库驱动之外，它还提供了数据库抽象访问的一些基本方法，包括 CURD 的 SQL 指令的抽象访问。查询的结果返回数据集对象 ResultSet，而 Dao 对象把查询的数据集从数据集对象又转换成了 Vo 对象或者 VoList 对象。最终的目的是为了让应用操作数据库更加方便有效。

在每个数据库驱动类库中，我们定义了几个方法，如果要增加其它的数据库驱动类库，请增加相应的同名方法。

必须方法：

Connect 连接数据库的方法

Query 查询方法



Execute 执行方法

getAll 取得查询数据集的方法

close 关闭数据库

free 释放查询结果

error 数据库错误方法

非必须方法

Commit 提交方法

Rollback 回滚方法

getRow 取得某条记录方法

next 记录指针下一条记录

escape\_string 字符安全过滤

另外注意数据库公共访问类 Db 定义了一些变量，这些在访问和操作数据库时候经常用到，不需要在数据库驱动类库中单独定义，他们是：

```
// 数据库配置

var $DbType      = NULL;      //数据库类型
var $DbVersion   = NULL;      //数据库版本
var $AutoFree    = False;     //是否自动释放
var $AutoCommit  = True;      //是否自动提交
var $Debug       = True;      //是否显示调试信息
var $Pconnect    = True;      //是否使用 pconnect
var $RsBuffer    = False;     //是否缓存查询结果 暂时没用

// 数据库连接 ID 和当前查询 ID
var $LinkID      = 0;
var $QueryID     = 0;

// SQL 查询指令
```


```
var $QueryStr = "";

// 查询结果

var $Result = NULL;           //当前结果
var $ResultSet = NULL;       //结果数据集
var $ResultType = 1;         //0 返回数组 1 返回对象
var $LastInsID = NULL;       //最后插入 ID
var $NumRows = 0;            //返回或者影响记录数
var $NumCols = 0;            //返回字段数
var $QueryTimes = 0;         //查询次数
var $WriteTimes = 0;         //写入次数
var $TransTimes = 0;         //传送次数 用于 rollback
var $error = "";             //错误信息
```

如果直接用 Db 类来操作和访问数据库的话，需要在数据集类库 ResultSet 中获取记录，ResultSet 中的记录格式可以通过设置 Db 类的 ResultType 变量(0 返回数组 1 返回对象)。可以用 getRow(\$Row) 方法来取得某条记录，也可以用 getRange (\$offset,\$length)方法来获取某个范围内的记录。(如果直接通过数据库驱动类库来访问和操作数据库的话，还可以使用 next 方法和 result 来获取数据集)。Db 类提供了 Find、Add、Remove 和 Save 四个抽象的操作方法。还提供了切换数组和对象返回类型的方法 getArrayIterator 和 getObjectIterator。

如果使用 Dao 对象来操作数据的话，返回的是 Vo 对象或者 VoList 对象，针对 VoList 对象可以使用 get 方法和 range 方法（继承父类 ArrayList 的方法）来分别获取某条记录和一定范围内的记录。

 注意：如果要考虑记录的分页显示问题，可以从获取某个范围内的记录方法着手分析，使用 ResultSet 的 getRange 或者 VoList 对象的 range 方法都可以。

### 3.9.4 工具类库

工具类库有很多，主要是提供应用开发时候的基础类库支持，在架构的数据模型实现的时候，用到的比较重要的几个类库是：ListIterator、ArrayList 和 HashMap。

因为 `ResultSet` 数据集对象由 `ListIterator` 派生而来, `Vo` 对象、`VoList` 和 `Dao` 对象的操作都需要 `HashMap` 和 `ArrayList` 的支持, 而 `HashMap` 和 `ArrayList` 也都派生自 `ListIterator` 类。

工具类库可能是 `FCS` 基类库中最庞大的类库, 目前实现的大部分是基础类和抽象类, 以后可能会添加一些更多的具体类, 目的当然是提高应用开发的效率和提高类库的可重用性, 让应用开发看起来更加简单。当然, 您还可以在应用类库中单独添加自己的工具类库, 如 `com.liu21st.App1.util`。

### 3.9.5 异常处理类库

系统的异常处理都派生自 `FCSEException` 基础类, 暂时系统并没有扩展其它异常类型, 因为兼容性的需要, 所以无法判断要抛出的异常类型, 但是在进行应用开发的时候, 可以扩展模块的异常处理, 在模块内出错均抛出该模块的异常即可。也可以直接抛出系统异常。

由于照顾到 `PHP4` 的异常处理问题, 所以, 如果在 `PHP` 环境下作业, 不能通过 `throw` 某个异常的方式来抛出异常, 也不能用 `try` 和 `catch` 来捕获异常。只能通过 `throwException` 方法来手动抛出异常。因此请根据你的项目要求来使用不同的异常处理方式。而且 `throwException` 默认只能抛出系统的异常, 如果你扩展了某个异常处理类, 需要在 `throwException` 方法中抛出, 请使用 `$type` 参数来指定要抛出的异常类型 (异常处理类的名称) 或者添加单独的抛出异常方法。

例如, 抛出 `UserException` 异常的方法

```
throwException('用户信息错误','UserException');
```

## 第4部分 模板指南

### 4.1 模板概述

FCS 内置的模板引擎提供了丰富的模板标签功能，尤其是对 **Vo** 和 **VoList** 的输出支持。为页面的编辑和列表提供了极大的方便。而且模板引擎具有非常强的可扩展性，为以后添加新的模板标签元素实现了可能。

### 4.2 模板结构

FCS 的模板结构和系统是 100%分离的，可以独立制作和预览，而没有任何依赖，除了模板标签的输出之外。模板的结构请遵循下面的规范，以确保模板读取正常。

| -Tpl 模板根目录

| --default 主题目录（主题目录下面存放网站目录，对应网站的入口文件）

| -----admin 网站目录 1

| -----index 网站目录 2（每个网站目录下面存放模块目录，对应模块的应用类库）

| ----- index 模块目录 1（每个模块目录下面存放模板的操作方法文件）

| ----- index.html index 操作模板文件

| ----- add.html add 操作模板文件

| ----- blog 模块目录 2

| ----- public 公共目录（模板的外部资源文件引用通过该相对路径访问）

（以下目录可以自行修改，不影响模板读取）

| ----- images 建议图像目录

| ----- css 建议样式目录

| ----- js 建议脚本目录



简单的说，模板文件的访问路径是：

模板目录 / 模板名 / 项目名 / 模块名 / 操作方法名

例如

Tpl/default/index/fcs/index.html index 项目的 fcs 应用模块中的 index 操作方法

Tpl/default/admin/user/add.html admin 项目的 user 应用模块中的 add 操作方法

注意：只有需要显示的 Action 方法才需要定义模板文件

每个项目下面都有个 public 目录放置这个项目的一些公共资源文件

（事实上在模板名下面也有一个 public 目录，但是这个目录是供系统本身存放模板文件和调用的）



注意：不要定义名称为 public 的项目和模块，public 已经被 FCS 系统作为保留用途。

## 4.3 模板定义

在模板文件中如果要调用外部资源（如图片、样式表和脚本文件等），请使用相对路径，这样做的作用是让模板制作者在制作的时候可以正常预览，而系统也可以正常读取。资源文件的统一路径使用 public 目录，例如

//样式调用

```
<link rel='stylesheet' type='text/css' href='../public/css/style.css'>
```

//图像调用

```
<IMG SRC='../public/images/banner.jpg' WIDTH='468' HEIGHT='60' BORDER='0' ALT=''>
```

//脚本调用

```
<script type='text/javascript' src='../public/js/common.js'></script>
```


注意，系统在解析模板的时候会替换所有../public 的路径，所以，如果你确实要在页面中输出../public 字符串，请在其间添加空格，如 ../ public 否则会被自动替换，需要引起注意。

在做应用类库开发的时候，所有需要输出的操作都需要有一个模板文件，模板文件的编码请按照配置文件中 TEMPLATE\_CHARSET 的设定，不需要考虑输出格式，页面输出的时候系统会自动转换成

OUTPUT\_CHARSET 的设定编码输出，也包括会把模板文件中的编码设定替换。如果 TEMPLATE\_CHARSET 的设定为 GB2312，那么模板文件中应该写成

```
<META http-equiv='Content-Type' content='text/html; charset=GB2312'>
```


在输出的时候，系统会自动替换 GB2312 为 OUTPUT\_CHARSET 的设定编码。

 注意：如果您的空间不支持 iconv 模块或者 mbstring 模块的话，那么请把模板和输出编码全部采用同一种编码格式。

模板文件的后缀在项目配置中可以配置。

## 4.4 模板标签

模板文件一般包含一些模板标签（当然，你完全可以不借助模板标签来实现，而直接使用 PHP 的语句来输出变量），如果你使用其它模板引擎作为页面输出，那么请遵守相关的模板标签规定。

 您一样可以使用诸如 Smarty 或者 TinybutStrong 之类的模板引擎。但是在使用之前，您可能要做一些模板的设置工作。然后按照手册提供的调用方法使用即可，如果你确信要这么做的话，上面的模板结构可能对你没有帮助，并且要作出牺牲某些特色标签的思想准备。

FCS 系统的模板标签以 { 和 } 作为开始和结束标识（如果需要更改请在配置文件中修改），并且在开始标记紧跟标签的定义，如果之间有空格或者换行则被视为非模板标签直接输出。下面列出一些目前版本实现的模板标签的使用说明。

## 4.5 模板变量

内置模板引擎对模板变量的支持相当充分，除了可以输出正常变量外，还可以输出特殊变量和对输出变量使用函数定义（并支持多个函数）。

模板变量的标签格式为

格式：{**\$varname** | **function1** | **function2=arg1,arg2,###** }

说明：

{ 和 \$ 符号之间不能有空格 ， 后面参数的空格就没有问题

###表示模板变量本身的参数位置

支持多个函数，函数之间支持空格

支持函数屏蔽功能，在配置文件中可以配置禁止使用的函数列表

支持自定义函数，以后打算封装一些常用格式函数

支持变量缓存功能，重复变量字串不多次解析

使用例子：

```
{ $webTitle|md5| strtoupper | substr=0,3 }
```

```
{ $number|number_format=2 }
```

如果在应用 Action 中需要输出某个变量，使用下面的方法：

```
$tpl = new Template();
```

```
$tpl->assign('name','value');
```

系统只会输出设定的变量，其它变量不会输出，一定程度上保证了变量的安全性。

除了常规变量的输出外，模板引擎还支持系统变量和系统常量、以及系统特殊变量的输出。它们的输出不需要作变量设定。

支持系统变量（依然支持函数使用和大小写、空格），以 FCS.打头，如

```
{ $FCS.server.script_name } //取得$_SERVER 变量
```

```
{ $FCS.session.session_id|md5 } //版本$_SESSION 变量
```

```
{ $FCS.get.pageNumber } //版本$_GET 变量
```

```
{ $FCS.cookie.name } //版本$_COOKIE 变量
```

系统常量

```
{ $FCS.const.__FILE__ }
```

```
{ $FCS.const.MODULE_NAME }
```

特殊变量 ， 由 FCS 系统定义的常量

```
{ $FCS.version } //版本
```

```
{FCS.now } //现在时间
{FCS.template|basename } //模板页面
{FCS.LDELIM } //模板标签起始符号
{FCS.RDELIM } //模板标签结束符号
```

## 4.6 模板注释

模板支持注释功能，该注释文字在最终页面不会显示，仅供模板制作人员参考和识别。

格式：`{/* 注释内容 */}` 或 `{// 注释内容 }`

说明：在显示页面的时候不会显示模板注释，仅供模板制作的时候参考。

注意{和注释标记之间不能有空格。

## 4.7 公共模板

当页面需要包含公共文件的时候，可以通过下面的模板标签

格式：`{ include:Filename }`

说明：Filename 表示公共文件的名称（不包含后缀，因为模板文件后缀为可配置），Filename 默认在当前目录下寻找，但是完全支持相对路径访问，例如，下面的格式都是正确的。`{ include:header }` 和 `{ include:../public/header }`。该标签可以出现在模板页面的任何位置，也就是说可以包含除了头部文件和尾部文件之外的一些公共文件，就看网站页面的设计了。建议的方式是首先让美工写在一个页面文件中，然后调试完毕后在分成多个文件来引用，因为一旦使用引用标签后，模板页面就不能直接浏览到实际的效果了。加载公共模板文件后，模板引擎会重新对该页面中的模板标签进行解析，有意思的是你还可以在公共模板中再次包含公共文件，但是一定要注意不能循环包含。例如，在 header.html 文件中包含了 menu 文件

```
{include:menu}
```

在 index.html 文件中则包含了 header 和 footer


```
{include:header}
```

这里是首页的内容

```
{include:footer}
```



在访问 `index` 操作方法的时候,模板首先读取 `index` 文件,并开始解析 `include:header`,在解析 `header` 文件的过程中又遇到 `include:menu` 标签,又开始解析,解析完成后再解析 `include:footer` 标签,在经过几层的嵌套包含解析后 `index` 文件最终被解析成一个缓存模板文件。

 注意: 由于模板解析的特点,从入口模板开始解析,如果公共模板有所更改,模板引擎并不会重新编译模板,除非缓存已经过期。如果遇到比较大的更改,您可以尝试把模块的缓存目录清空,系统就会重新编译,并解析到最新的公共文件了。

## 4.8 对象标签

注意到模板变量的输出并没有定义对象的属性输出,由于性能的关系,FCS 系统没有对对象的属性输出提供支持,但是有一种对象除外,就是数据模型 `Vo` 对象,考虑到在进行数据操作的时候,经常需要输出某个数据表的一些属性,以供查看和修改,虽然,利用原来的模板变量输出也可以进行对象的输出,但是过程会相对麻烦些,因为要把各个数据表的字段逐个赋值给输出变量,势必存在对 `Vo` 对象的遍历过程,甚至如果有多个 `Vo` 对象要输出那就更加麻烦了。

因此,FCS 系统特别针对 `Vo` 对象做了输出处理,使得在模板中输出 `Vo` 对象显得轻而易举。

格式: **{Vo: VoName| key }**

说明: 在页面中输出名称为 `VoName` 的 `Vo` 对象的属性 `key` 的值

如果在应用 `Action` 中需要输出某个 `Vo` 对象,使用下面的方法:

```
$tpl = new Template();  
$UserVo = new UserVo($Map);  
$tpl->assignVo('user',UserVo );
```

## 4.9 列表标签

如果要输出一个数据集呢,如果使用对象标签岂不是累的要死,呵呵~所以,系统专门为数据集的列表输出定义了列表标签。其实,数据集就是包含多个 `Vo` 对象的 `VoList` 对象,

格式: **{VoList: VoListName| Content }**

说明：在页面中输出名称为 VoName 的 VoList 对象

其中 Content 中可以直接输出 Vo 的属性名称，方法是使用 Vo 属性名称作为变量，VoList 中的变量不用再添加标签开始和结束标记

如下面的例子：

```
{voList      :      user      |      <H4><A      HREF="?a=edit&id=$id">$username</A>      <A  
HREF="?a=edit&id=$id">编辑</A> <A HREF="?a=delete&id=$id">删除</A></H4> };
```

如果在应用 Action 中需要输出某个 VoList 对象，使用下面的方法：

```
$tpl = new Template();  
$dao = new UserDao();  
$userList = $dao->findAll();  
$tpl->assignVoList('user',$userList);
```

## 第5部分 附录

### 5.1 文件列表

Fcs.php	框架系统公共文件
Functions.php	公共函数库
Config.ini.php	系统配置文件
Index.ini.php	index 项目配置文件
Zh-cn.php	简体中文语言包文件
Copyright.txt	版权说明文件

## 5.2 类库列表

### FCS.core 核心类库

Base.class.php	FCS 系统基类
App.class.php	应用控制类
Action.class.php	模块操作基础类
Vo.class.php	数据对象基础类
Dao.class.php	数据访问基础类
Template.class.php	内置模板引擎类

### FCS.db 数据库访问类库

Db.class.php	数据库公共类库
Db_mysql.class.php	mysql 数据库驱动类库
Db_mysqli.class.php	mysqli 数据库驱动类库
Db_pgsql.class.php	pgsql 数据库驱动类库
Resultset.class.php	数据集类库

### FCS.util 工具类库

Listiterator.class.php	模拟 ListIterator 类
ArrayList.class.php	模拟 ArrayList 类
HashMap.class.php	模拟 HashMap 类
Stack.class.php	模拟 Stack 类
Log.class.php	系统日志类

### FCS.Exception 异常类库

FCSException.class.php	FCS 系统异常基础类
------------------------	-------------

## 5.3 函数列表

Halt	错误输出方法
ThrowException	抛出异常方法
autoCharSet	自动转换编码
varFilter	变量安全过滤
__autoLoad	自动加载类库方法, PHP5 有效
includeOnce	优化的 include_once 方法
requireOnce	优化的 require_once 方法
import	引入类库方法
using	import 方法的别名
property_exists	检测对象的属性方法, PHP4 有效
array_combine	合并数组方法, PHP4 有效
systemOut	调试输出方法