SeaPHP 开发架构使用手册

Author： JerryLi（lijian@dzs.mobi）

Ver:1.0.0 （2014-03-05）

一、架构介绍

1.1、系统特性

依赖于 PHP5.2 环境。

SeaPHP 架构是一种轻量化可配置的高性能架构，并且采用了面向对象的设计思路进行规划。它将 PHP 的设计模式分为四部分，分别为：系统内核、扩展框架、日志区、工作区。与其他常用的 PHP 设计框架的最大区别是它的内核抽象层次更高，通过内核来动态的加载需要使用哪个框架。每个框架有很清晰的目录结构，互不重叠互不干扰，让使用者很容易上手，并且能够很容易的进行扩展。

架构的内核所做的功能是管理整个系统中所有类的动态加载，与最底层的公用功能封装。架构通过引入了动态配置文件的方式，对动态设置信息进行分离，因此仅需几分钟的时间，就能重新配置与发布一个基础系统，而不用修改内部的任何核心代码，大大增加系统的健壮性于易用性。

架构的扩展框架设计，可让高级的开发者提供了用于解决某些特殊问题的框架扩展接口。本系统现在提供了两个框架，数据库设应用层框架与 WebService 应用框架可方便的解决实际问题。

架构的工作区设计目的，是将核心代码与项目的业务逻辑进行分离，作为项目的业务开发者，只需要在工作区编写相应的业务逻辑，不需要去改动各自框架的核心代码。如此当系统发布到线上系统或者更新业务逻辑时，只需要将工作区的文件更新到服务器上，或者更新对应的配置文件，从而降低系统更新时的风险。

SeaPHP 架构内所有 php 文件的开发注释都按照规范编写，如果你使用 Zend Studio 9 这类的带代码提示的 IDE 环境进行开发，则可以完全脱离使用手册，就能进行快速的开发，非常适合团队的研发与管理，架构的设计干净简洁。

1.2、架构的目录结构

系统架构的主目录名是 SPWF，它存放在 Web 项目的二级目录中，以减少对项目目录结构的干扰。

SPWF 目录下只有 4 个目录区域，分别为 core:内核区、extend:扩展框架区、log:日志区、workgroup:业务逻辑文件开发环境区。因此 SeaPHP 天生具备极佳的可扩展性。并且让各扩展框架功能框架之间有很好的隔离。每个框架维护自己的动态加载文件与类库文件，不会对内核产生干扰。

core 目录区中，只包含系统的运行环境所依赖的最核心部件，以及相对于全局的依赖性最强的公共函数库，不含任何业务逻辑。因此开发者几乎不需要需修改 core 中的除配置文件以外的任何文件。

extend 目录区中，每个二级目录都是一个设计框架或者通用数据模块的隔离区，他们各自维护自己的配置文件与动态加载类，因此各框架与通用模块之间互不干扰，具有很强的低耦合，高内聚的特性。

架构的目录结构：

project （Web 项目根目录）
- SPFW （SeaPHP 架构根目录）
- core （架构内核目录）
  - config （架构的环境配置目录）
  - lib （内核库）
    - base （内核库-基类）
    - final （内核库-终态类）
    - sys （内核库-系统类库）
  - runtime （内核运行时目录）
- extend （扩展框架目录，每中框架为一个二级子目录）
- log （日志区）
- workgroup （开发者工作区）

1.3、命名规范

良好的命名规范能让开发出来的代码更加优雅与减少错误的发生。SeaPHP 遵从的命名规范是：

- 类文件：尽量以大写 C 开头，.class.php 格式结尾，每个类文件中只能定义一个类，类名要与文件名相同。
- 接口文件：必须以大写的 I 开头，.php 结尾，每个接口文件中只能定义一个接口，接口名与接口文件名必须相同。
- 变量定义：类内的全局变量必须以$m 开头。每个变量的前导字母必须表示出变量的类型且必须小写，变量名称首字母大写（例如:$maGet:全局数组变量名称为 Get，$bIsRun：布尔型变量变量名称为 IsRun）。
- 类内常量定义：与变量定义规范相同，只是变量名用大写单词之间用’_’分隔（如 const msFIELD_NAME）。
- 函数定义：所有函数尽可能按照：[动词+名词]这样的格式来命名，并且动词必须小写，名词首字母必须大写（例如：获取变量值 function getName()、保存日志 function saveLog、显示历史记录 function showHistory()）。
- 类内成员变量与成员函数定义：必须标识出 public / protected / private 这 3 中访问权限。

1.4、执行流程

图

1.5、入口文件的配置

本架构的所有设计框架均使用单一入口的文件的设计模式，通过同一个 URL 地址来访问所有 Web 服务。

```
<?php
define('SEA_PHP_ROOT', '/SeaPhp/'); //Web 项目的根(如项目位于二级目录时才需要此设置)
require './SPFW/sea_php_init.php'; //SeaPHP 环境的引入文件
_dbg('SeaPHP say:Hello world.');
?>
```

[Code 1.5.1 入口文件]

如[Code 1.5.1]所示，第二行 require './SPFW/sea_php_init.php'; 就是 SeaPHP 架构的引入语句，

引入架构后，就可以使用架构提供的所有服务，或者启动架构内的某个扩展框架。

SeaPhp 架构 Web 根，如果在二级目录下，则需要在入口文件中加入这句定义，并修改其根目录值（注意：前后都必须加上'/'）*define('SEA_PHP_ROOT', '/SeaPhp/');*

SeaPhp 架构的根目录的名称，如果修改了架构根目录的名称，则需要在入口文件中加入这句定义，并修改其根目录值（注意：前后都不要出现'/'）*define('SEA_PHP_SPFW_ROOT', 'SPFW');*

SeaPhp 架构的本机访问绝对路径，如果使用命令行方式执行(非 Web 代理方式执行)，则需要加入这句定义，并修改本机的绝对根路径（注意：末尾不要带 '/'）*define('SEA_PHP_MACHINE_ROOT', 'E:/webroot');*

1.6、架构内核的配置

/SPFW/core/config/ 目录存放了内核配置文件：

- autoload.cfg.php 为动态加载类的配置，一般不需要去修改由内核管理员维护。
- environment.cfg.php 为架构的系统环境配置文件，其中'show_debug_info'参数如果是线上系统，请设置为 false。
- extend_framework.cfg.php 为架构的扩展框架与公共扩展模块的入口类加载文件配置。一般不需要去改动。

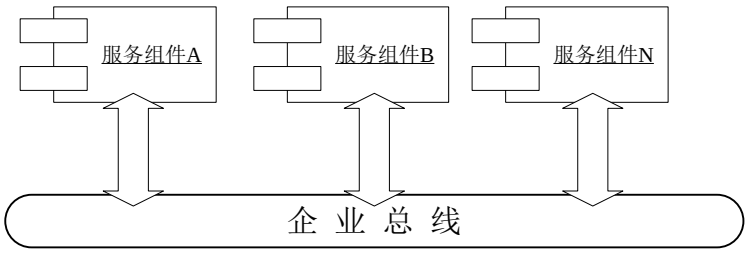
由内核的配置文件可以看出，要配置一个 SeaPHP 的新开发环境，非常简单。

二、架构内的扩展框架介绍

2.1、WebService 开发框架介绍

2.1.1 框架特性

WebService 是如今基于 SOA 应用软件架构设计中最常用的开发模式，其部署结构如下所示：



[Img 2.1.1 SOA 架构示意图]

上图所示，说明了 WebService 的所有特性：

- 使用唯一入口访问所有服务组件；
- 服务组件动态寻址；
- 服务组件可随意的增加与缩减（具备弹性化业务设计的逻辑理念）；

WebService 框架实现了企业总线（即单一入口）、服务组件的维护管理、组件服务的发现反射机制，因此他具备了 WebService 的所有机制，你可以通过简单的修改配置文件中的参数来快速部署一个 WebService 服务。同时开发服务组件的整个过程会非常的简单，支持多人的团队协同开发。对于服务组件的调用者，能通过服务协议的发现机制，快读的了解到接口的协议与使用信息，并且能够支持在线调试与测试（无需写一行 Ajax 或者写 httppost 方法就能对接口进行调试）。

WebService 框架支持的传输协议，可同时支持 [XML | JSON] 的输入与输出，也支持 URL 的 GET 方式（对输入格式有特殊要求）的输入与[XML | JSON]的输出。因此他有极强的兼容性。

对于 WebService 服务组件 API 的开发者来说，整个开发过程会非常愉悦与畅快。你无需周旋与 Xml 与 Json 的解析处理，这都由系统帮你完成。你可以将尽可能多的精力放在对业务逻辑的实现上（大大提高你的工作效率），并且在编写服务组件的同时，你只需要按规范实现 ProtocolView 接口的抽象函数，系统就为你自动生成完善的接口协议文档，通过一种反射机制实现实时的发布（支持 Web 端的接口在线测试）。

2.1.2 框架的目录结构

WebService 框架目录的内核位于 [SPFW/extend/webservice] 中、服务组件位于 [SPFW/workgroup/webservice/package] 、服务组件的访问日志位于 [SeaPhp/SPFW/log/extend/webservice]

目录结构：

SPFW/extend/webservice/ WebService 内核目录
- config/ WebService 环境配置文件
- lib/ WebService 框架类库文件
- runtime/ WebService 运行时文件(入口类)
- template_protocol/ WebService 协议视图模板

SPFW/log/extend/webservice WebService 服务组件的访问日志

SPFW/workgroup/webservice/package WebService 工作区（服务组件业务逻辑）

2.1.3 入口文件的配置

WebService 入口文件需要包含下面这些参数：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
define('SEA_PHP_ROOT', '/SeaPhp/');
require './SPFW/sea_php_init.php';
//如果 get 参数存在 protocol_type=json，则使用 json 协议，否则默认使用 xml 协议
CExtendManage::Run(new CWebService());
?>
```

[Code 2.1.3.1 WebService 入口文件]

如[Code 2.1.3.1]格式，编写好 WebService 的服务单一入口文件，存放在网站根目录上（或者是任意的目录，注意第三行的路径设置），你的服务访问者(如文件名问 webservice.php)就可以通过 http://website/webservice.php 来访问你所提供的 WebService 服务。

WebServiceProtocolView 是服务组件的发现服务，可以通过它看到所有服务组件的协议约定与使用介绍，入口文件需要包含下面这些参数：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
define('SEA_PHP_ROOT', '/SeaPhp/');
require './SPFW/sea_php_init.php';
CExtendManage::Run(new CWebServiceProtocolView());
?>
```

[Code 2.1.3.2 WebServiceProtocolView 入口文件]

如[Code 2.1.3.2]格式，编写好 ProtocolView 的服务单一入口文件，存放在网站根目录上（或者是任意的目录，注意第三行的路径设置），你的服务访问者(如文件名问 ProtocolView.php)就可以通过 http://website/ ProtocolView.php 来查看你在 WebService 中提供的所有的服务组件接口信息，并且能够在线调试这些 API 接口。

2.1.4 运行环境配置

需要使用 WebService 服务与开发服务组件时，必须对配置文件进行设置。配置文件目录位置在：[SPFW/extend/webservice/config]。

- 配置文件[autoload.cfg.php]不需要修改。
- 配置文件[environment.cfg.php]是框架的环境配置，根据需要进行设置，文件内对每条配置都有详细的说明。
- 配置文件[auth_protocol.cfg.php]是 Protocol view 的授权访问验证帐号配置。它的功能是在编写服务组件时，开发人员可以设定本接口能让哪些人看到其接口协议的详细内容。只有当用户在[协议视图页面]右上角切换帐号后，如果这个帐号有权限查看此接口的信息他就查看，否则权限不够时将看不到这个接口的协议信息。
- 配置文件[package_access_pwd.cfg.php]用于服务组件访问时的 checksum 校验检查的公钥密码，每个根包都必须设置这个密码。

2.1.5 服务接口的使用规范

WebService 的服务接口访问时，通过一种类似 Java 包管理的机制，传入 package 与 class 来对企业总线上的各种服务组件进行访问，所以每个接口文件必须包含 package 与 class 两个参数节点。同时为了防止非认证的用户访问服务组件，每次访问时还需要包含 checksum 节点参数，如果访问者没有得到授权的公钥，则 WebService 会拒绝他的访问请求。

根据如上的规范介绍，我们通过一个 XML 访问包的范例来讲些这个访问规范。

```
<?xml version='1.0' encoding='utf-8'?>
<boot>
    <package>develop.test</package>
    <class>GET_USER_INFO</class>
    <checksum value='校验码 md5(32 位)' unix_timestamp='Unix 新纪元(格林威治时间 1970 年 1 月 1 日 00:00:00)到当前时间的秒数'/>
</boot>
```

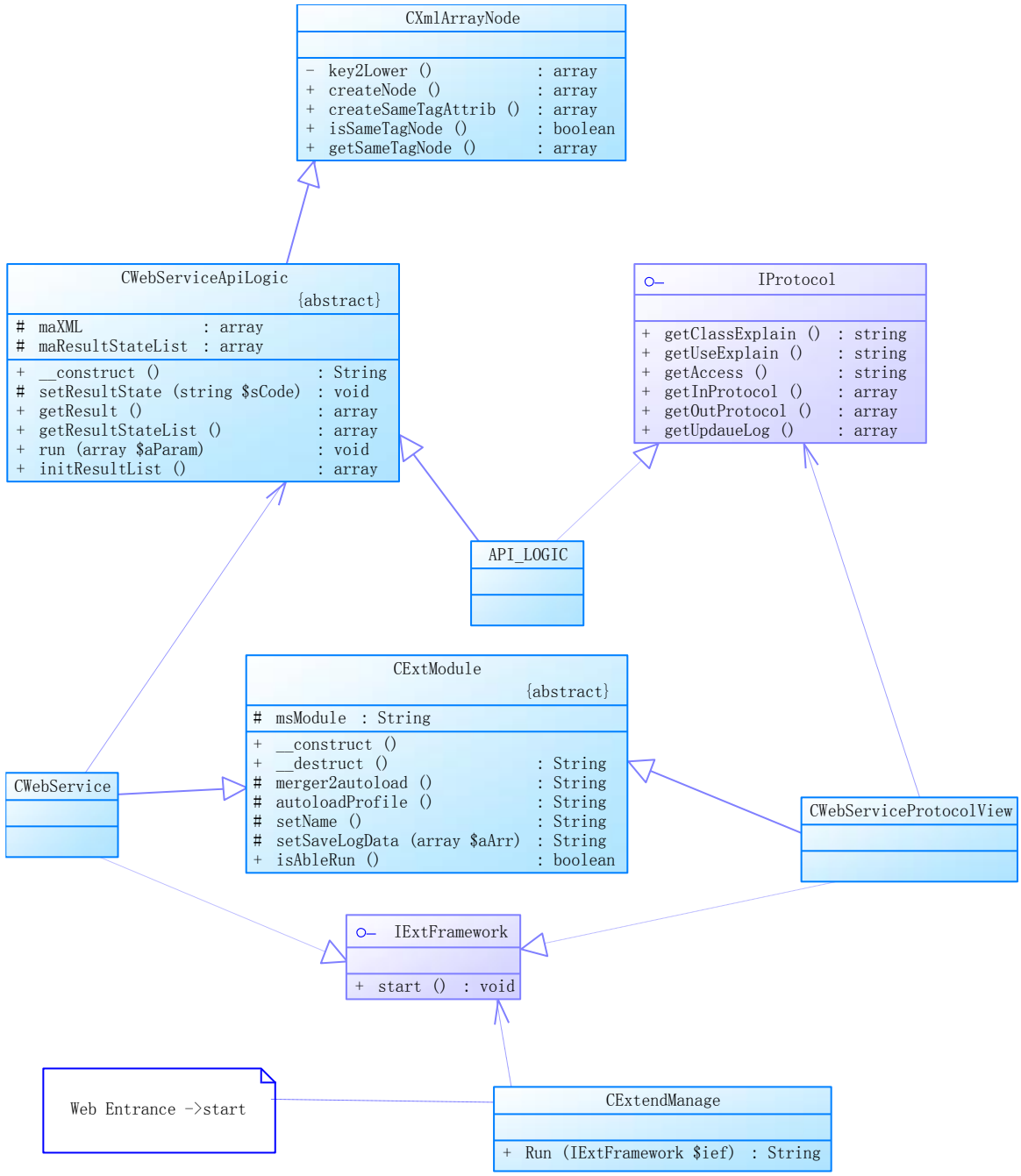
[Code 2.1.5.1 服务组件的访问规范]

[Code 2.1.5.1]所示，package 为服务组件的包地址，class 是服务组件的业务逻辑类，最后是 checksum 节点的 value 属性是根据算法约定生成的校验码，unix_timestamp 是 Unix 新纪元(格林威治时间 1970 年 1 月 1 日 00:00:00)到当前时间的秒数(php:time() | java:System.currentTimeMillis()/1000 | JavaScript:Math.round(new Date().getTime()/1000))。这个是 WebService 企业总线为客户端提供访问请求服务的最基本信息，所以对任意一个服务组件进行访问时都需要包含这 3 个节点信息。

package 的命名规范：可以使用大小写（但是目录名必须与 package 的大小写完全相同），不同层次的包层次结构，使用'.'进行分隔。例如：develop.test

class 的命名规范：必须使用大写（非强制，但是希望能按照这个规则执行），类名与类文件名称必须相同，一个类文件中只能包含一个类。

2.1.6 框架的类关系图



2.1.7 XML 结构数组的映射原理(*)

引入这个特殊的结构数组的是本框架内最核心的内容，开发者必须掌握这个映射原理。掌握这个映射关系后，就能为你节省大量的时间与精力，投入开发服务组件的工作中。

Xml 的最基本的结构可分解为：tag, attribute, content 三部分。

tag 的关系有 3 种：父子的层级关系；同级的兄弟 tag 关系；同级的同名兄弟关系。

```
<root>
<tag1>content1</ tag1>
<tag2 key1=”val1” key2=”val2”/>
<tag3 key3=”val3”/> content3</tag3>
<tag4>content4</tag4>
<tag4>content5</tag4>
<info>
```

```
    <row key4=”val4” />
</info>
</root>
```

[Code 2.1.7.1 XML 结构数据包示例]

PHP 的数组是以 Hash 方式存储，因此参照[Code 2.1.7.1]中 tag 的层级关系很容易被表示，如[‘info’][‘row’]。另外一个 tag 节点可能同时存在两个参数 content:内容 与 attribute:属性，因此可以用一个特殊标志来表述 conetnt 与 attribute，[tag1][‘C’]表示 content，[tag2] [‘A’]表示 attribute。下面通过一个表来展示这种映射关系。

编号	XML 节点	PHP 数组
1	< tag1>content</tag1>	[“tag1”][“C”]= “content1”;
2	<tag2 key1=”val1” key2=”val2”/>	[“tag2”][“A”]= array(“key1”=>”val1”, “key2”=>”val2”);
3	<tag3 key3=”val3”/> content3 </tag3>	[“tag3”][“C”] = “content3”; [“tag3”][“A”]=array(“key3”=>”val3”);
4	<tag4>content4</tag4> <tag4>content5</tag4>	[“tag4”]= array(array(“C”=>“conteng4”), array(“C”=>“conteng5”));
5	<info> <row key4=”val4” /> <info>	[“info”][“row”][“A”]= array(“key4”=>”val4”)

[表 2.1.7.1 XML 与 PHP 数组的映射关系表]

请熟练掌握这个映射关系，后面在对服务组件接口的开发中，将会用到这个 XML 数组的使用。

2.1.8 服务组件 API 接口的开发

WebService 中的 API 接口逻辑文件存放于[SPFW/workgroup/webservice/package]。

API 接口逻辑类文件的路径就是[2.1.5 节]中说到的 package 这个节点参数，此参数的值对应的就是在[SPFW/workgroup/webservice/package]下的一个目录接口，所以每个包都有一个根包。例如 :package=develop, class=GET_LIST，那么 API 接口逻辑类文件的路径就是[SPFW/workgroup/webservice/package]，类文件是 GET_LIST.php，类名是 GET_LIST，通过系统总线就能把从客户端发送过来的 xml 解析城 XML 结构数组，送到 API 接口逻辑类文件中。

类文件的编写，首先看一个 API 接口类文件的示例：

```
<?php
/**
 * API 服务接口类<br />
 * 备注：在这个类中编写业务接口服务的业务逻辑，使用时必须继承 CWebServiceApiLogic 类与 IProtocolView 接口。<br />
 * 如果不继承 IProtocolView 接口，那么当前类将不具备 Protocol View 的反射调用。
 * @access public
 * @final
 */
final class GET_USER_INFO extends CWebServiceApiLogic implements IProtocolView
{
    /**
     * 构造函数
```

```
     * @see CWebServiceApiLogic::__construct()
     */
    function __construct()
    {
        parent::__construct();
    }

    /*
     * @see CWebServiceApiLogic::initResultList()
    */
    protected function initResultList()
    {
        return array
        (
            '0000'=>'处理成功',
            '0010'=>'用户不存在',
        );
    }

    /*
     * @see CWebServiceApiLogic::run()
    */
    public function run($aParam)
    {
        $this->maXML['level'] = parent::createNode('admin');
        $this->maXML['code'] = parent::createNode('330104xxxxxxxxxxx');
        $this->setResultState('0000');
    }

    /**
     * @see IProtocolView::getClassExplain()
     */
    public function getClassExplain()
    {
        return    '获取用户注册信息';
    }

    /**
     * @see IProtocolView::getUseExplain()
     */
    public function getUseExplain()
    {
        return '返回用户的信息列表，只有注册用户才能看到这个信息';
    }

    /**
     * @see IProtocolView::getAccess()
     */
    public function getAccess()
    {
        return 'public';
    }

    /**
     * @see IProtocolView::getInProtocol()
     */
    public function getInProtocol()
    {
        $aXml = array();
        $aXml['name'] = self::createNode('待查询的用户名字(max:32)');
        $aXml['Family'] = self::createNode(null, array('mother'=>'妈妈名字', 'father'=>'爸爸名字'));
        return $aXml;
    }

    /**
     * @see IProtocolView::getOutProtocol()
```

```
        */
        public function getOutProtocol()
        {
                $aXml = array();
                $aXml['level'] = self::createNode('会员级别[manage|admin|guest]');
                $aXml['code'] = self::createNode('证件号码[max:32]');
                $aRow = array();
                $aRow[] = array('uid'=>'101', 'pname'=>'admin');
                $aRow[] = array('uid'=>'...', 'pname'=>'.....');
                $aXml['row'] = parent::createSameTagAttrib($aRow);
                return $aXml;
        }

        /**
         * @see IProtocolView::getUpdaueLog()
         */
        public function getUpdaueLog()
        {
                return array(array('date'=>'2013-06-02', 'author'=>'jerryli', 'memo'=>'接口创建'));
        }
}
?>
```

[Code 2.1.8.1 API 接口逻辑类文件编写示例]

[Code 2.1.8.1]中 GET_USER_INFO 类必须继承 CWebServiceApiLogic 基类,实现 protected function initResultList()与 public function run($aParam)两个抽象方法。

initResultList()这个函数中直接返回由 API 接口逻辑类中定义的返回状态值。

run($aParam)这个函数为 API 接口逻辑类的运行入口，当接口被适配器选中执行后，会通过 run()方法将客户端发送过来的请求参数解析为 XML 结构数组用$aParam 参数送进来。API 业务逻辑类接口完成处理后，将要返回给客户端的 XML 结构数组保存到$this->maXML 中（注意 run()函数结束前必须设置一个状态，如：$this->setResultState('0000'); 其中的’0000’状态代码就是在 initResultList()中设置的那个数组）。至此 API 的业务逻辑处理流程结束，后面的步骤会由系统接管，将数据送回给请求客户端。

2.1.9 服务组件的协议编写

[Code 2.1.8.1]中可以看到 GET_USER_INFO 还有这行代码 implements IProtocolView，他表示必须实现 IProtocolView 接口定义中的方法，这些方法能够实现 API 接口的发现服务（即反射）。只需通过 Web 浏览 API 协议接口视图界面，就能看到这个全面的了解此接口服务的入口与出口协议、使用介绍、访问权限等详细信息。这是由系统自动生成的接口协议文档，而要使用这个功能必须实现 IprotocolView 接口中定义的抽象方法。

IprotocolView 接口定义中需要在实现类中完成以下方法的定义（参考 Code 2.1.8.1 样例代码）

getClassExplain(): 类功能说明

getUseExplain(): 类如何使用说明

getAccess(): 类视协议视图的访问权限[public|protected|private]，权限从左侧到右侧逐步提高。用户访问权限的设置在 SPFW/extend/webservice/config/auth_protocol.cfg.php 中进行配置。

getInProtocol(): API 入口协议设定

getOutProtocol(): API 的出口协议设定

getUpdaueLog(): 接口维护日志

2.1.10 接口服务的预览效果（反射功能）

接口完成开发后，可通过协议预览页的方式对外提供协议接口使用的详细调用说明与调用方式，同时也提供了调试接口的工具。

预览文件的编写，在网站对因目录下创建文件 ProtocolView.php（当前的例子在网站的根下创建的协议预览文件）代码如下：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
define('SEA_PHP_ROOT', '/SeaPhp/');
require './SPFW/sea_php_init.php';
CExtendManage::Run(new CWebServiceProtocolView());
?>
```

[Code 2.1.10.1 ProtocolView.php 文件]

创建完文件入口后访问 http://localhost/ProtocolView.php 就能看到如下界面（如果需要修改页面的显示模板，可找到 SPFW\extend\webservice\template_protocol 目录下内的文件，直接进行修改），所有接口的信息都会自动组织发布。

Air-ID Web Service Protocol View

权限组:private 切换

接口根 **XML协议** JSON协议 使用介绍

Package: SAC 〉 Enterprise

Package list :

- NodeFunc
- Report

Class list :

- ADD_SAC
- CHANGE_ENTERPRISE_LOGIN_PWD
- GET_ENTERPRISE_INFO
- GET_GPS
- GET_SPACE_ADDR_INFO
- GET_SPACE_ADDR_LIST
- GET_SPACE_ADDR_PARENT_LIST
- SET_ENTERPRISE_BASE_INFO
- SET_ENTERPRISE_EXT
- SET_GPS
- SET_SPACE_ADDR_BASE_INFO
- SET_SPACE_ADDR_EXT_ACCOUNT
- SET_SPACE_ADDR_HEARD_INFO

路由信息:
Package: SAC.Enterprise **Class :** ADD_SAC

接口介绍

包 功 能：企业信息管理区
类 功 能：企业新增加一个信息点
使用介绍: 如果企业还有信标可分配点数，能够调用本接口增加一个信标
公 开 度：protected(内部开发者的私有接口，开发者权限可查看)

入口协议约定(Http Get方式访问)

package=SAC.Enterprise&
class=ADD_SAC&
token=通行令牌([fixed:11]&
checksum.value=校验码md5(32位)&
checksum.unix_timestamp=Unix新纪元(格林威治时间1970年1月1日00:00:00)到当前时间的秒数

使用Get的方式提交请求信息时，要注意URL的QueryString中每个'='后面的值必须要符合 [RFC1738] 规范对参数进行编码。 URL的总长度不可超过2048bbtye。
注意：当入口协议节点的层次只有一层且不含同名兄弟节点时，才能使用get方式访问。

入口协议约定(xml格式)

```
<?xml version='1.0' encoding='utf-8'?>
<boot>
    <package>SAC.Enterprise</package>
    <class>ADD_SAC</class>
    <token>通行令牌([fixed:11]</token>
    <checksum value='校验码md5(32位)' unix_timestamp='Unix新纪元(格林威治时间1970年1月1日00:00:00)到当前时间的秒
</boot>
```

出口协议约定(xml格式)

```
<?xml version='1.0' encoding='utf-8'?>
<boot>
    <result value='状态代码' msg='状态代码解释'/>
</boot>
```

result节点返回状态值说明

0000 => 处理成功
0001 => Token已失效
0002 => Token无效
0010 => 剩余可分配点数不够
0011 => 信标开通失败(系统错误，可再尝试一次)
999 => There is no post data.(不存在post数据)
901 => Received protocol packets can not be resolved.(收到的协议包无法解析)
910 => Invalid input.(无效输入)
911 => Missing package and class node value.(缺少package与class节点值)
912 => checksum value attribute node does not exist.(checksum节点的value属性不存在)
913 => package, class can not be empty.(缺少package或class节点)
914 => The checksum validation did not pass.(checksum校验未通过)
915 => The package access password is not set Web Service.(Web Service 中未设置这个package访问密码)
916 => API interface class not found.(api接口服务类未找到)
917 => API interface services no output result set.(api接口服务无输出结果集)
918 => checksum verification fails, the server exceeds plus or minus one hour time difference.(checksum校
通过,与服务器时差超出正负1小时)
919 => checksum unix_timestamp attribute node does not exist.(checksum节点的unix_timestamp属性不存在
920 => checksum node does not exist.(checksum节点不存在)

更新记录

2013-06-22：[jerryli] 接口创建

接口测试

手工测试API接口

[Imge 2.1.10.1 ProtocolView.php 文件预览效果]

2.2、数据库操作层开发框架介绍(db)

2.2.1 框架特性

数据库操作是 Web 应用的基础核心功能，本框架的特点是支持分布式数据库的访问特性，只需要调整配置文件的参数，就能实现数据库的读写操作分离，并且支持多个只读库动态连接均衡负载，提高并发性。

在兼容性方面，支持了数据库层驱动（DB Driver）与数据库操作(CURD)分离，因此可以根据服务器配置的不同需求，更换底层驱动，例如：mysql 操作库与 mysqli 操作库，未来还能增加更多的底层驱动。

数据库所有操作都抽象到 CDbCURD 类中，且支持最原始的 SQL 语句操作与易用性效能更高的链式操作对数据库进行访问。

独有的表对象访问逻辑，通过继承 CdbTable 类，将对表的访问操作以表名为单位封装到表对象中，极大提高数据库业务操作的代码重用性，实现将业务逻辑与数据库处理逻辑的清晰分离。

独特的数据库连接控制，在数据库连接创建后并不会立即去连接诶数据库，直到遇到 SQL 语句需要执行的时候才会去连接数据库。所以可以把数据库操作对象放在操作的最顶层创建。在其他类内部通过全局变量的方式获取对象的操作引用。

2.2.2 框架的目录结构

DB 层框架目录的内核位于[SPFW/extend/db/runtime/]中、数据库表对象类位于[SPFW/workgroup/db/table_object/]、DB 操作日志位于[SeaPhp/SPFW/log/extend/CDB/]、数据库连接配置类位于[SPFW/workgroup/db/dsn/]。

目录结构：

SPFW/extend/db/	数据库层内核目录
config/	CDB 环境配置文件
lib/	CDB 框架类库文件
runtime/	CDB 运行时文件(入口类)
dsn/	数据库连接对象配置文件目录
SPFW/log/core/extend/CDB	数据库访问日志
SPFW/workgroup/db	数据库表对象逻辑文件目录

2.2.3 配置数据库连接

数据库连接类文件存在在 SPFW/workgroup/db/dsn/中，可以在目录中看到一个参考文件 CDbCfgLocalTest.php，你可以复制一份此文件，并修改其名称（注意类名必须与文件名相同），然后编辑这个数据库连接配置文件。

数据库连接配置包含量读写库与只读库的连接配置，如果你的系统需要使用数据库的读写分离操作，需要配置 getR()函数中的内容，否则只需要配置 getRW()中的内容。

主库（读写库）的配置方式：

```
public function getRW()
{    return array
     (
          'host'        =>' localhost', /*DB host address*/
          'username'  =>'......', /*LoginUserName*/
          'pwd'        =>'......', /*LoginPassword*/
          'dbname'    =>'......', /*DateBaseName*/
          'port'        =>3306 /*link port*/
     );
```

```
}
```

[Code 2.2.3.1 CDbCfgLocalTest.php 代码片段]

从库（只读库）的配置方式：

```
//如果不存在只读库如下配置
public function getR()
{
	return null;
}

//如果存在只读库如下配置：
public function getR()
{
	return array
	(	//分布式只读库配置（如果存在多个，系统会随机选择一个连接）
		array('host'=>'', 'username'=>'', 'pwd'=>'', 'dbname'=>'', 'port'=>3306),
		array('host'=>'', 'username'=>'', 'pwd'=>'', 'dbname'=>'', 'port'=>3306),
	);
}
```

[Code 2.2.3.2 CDbCfgLocalTest.php 代码片段]

只读库可以同时配置多个连接记录如[Code 2.2.3.2]中的 getR()所示，可以设置 N 个数组配置对象。当需要进行分布式只读处理时，数据库连接管理类会随机选择一个只读库进行连接，以便随机分配并发流量。

数据库环境变量的配置：

```
public function getEnvironment()
 {	return array
	(
		'prefix'	=>'', /*表前缀*/
		'table_upper_lower'	=>'lower',  /*表名强制大写[upper:强制大写|lower:强制小写|intact:保持原样]*/
		'db_driver'=>'mysqli' /*数据库驱动[mysql:兼容新最好|mysqli:性能最好]*/
	);
}
```

[Code 2.2.3.3 CDbCfgLocalTest.php 代码片段]

代码[Code 2.2.3.3]中表前缀为可选项（不需要可以为空）；表名强制大小写如果不使用需设定成”intact”；数据库驱动暂时只提供了 mysqli 与 mysql 两种驱动，如果需要用 oracle 或者 MsSqlServer 等，可以参照[extend/db/lib/CDbDriverMySql.class.php]继承 CDbDriver.class.php 中的 CdbDriver 类，并实现所有抽象方法。

2.2.4 数据全局环境变量配置

在[extend/db/lib/environment.cfg.php]中设定了数据库全局环境变量，如自动记录操作日志，数据库默认字符集，默认读写库的操作，与数据库表对象的存放路径等。详细内容直接参考文件中的注释进行修改。

2.2.5 数据库操作（初级）

对于初次使用的用户无需学习 CURD 的链式操作方式，可将原有的 SQL 语句快速移植到本系统的数据库层框架。

首先在入口文件中实例化一个数据库操全局操作对象：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
require './SPFW/sea_php_init.php';
$odb = new CDB('CDbCfgLocalTest');
?>
```

[Code 2.2.5.1 代码片段]

在[Code 2.2.5.1]中，$odb 就是一个全局的数据库操作对象，'CDbCfgLocalTest'是 2.2.3 中所讲到的某个数据库的连接配置文件。后面的所有数据库操作都通过$odb 进行操作。如果在类文件中操作这个$odb，需要使用 global $odb; 的方式获取全局数据库操作对象的引用。

基本数据库操作：

SELECT 操作：

以下所有函数的形参中

bool $bDBWR 设置为 null 表示按照 dsn 中的默认配置的读写方式进行读写分离操作。

string $sSql 是查询的 SQL 语句；

- $odb->db()->select($bDBWR=null, $sSql);

  返回整个 SQL 语句的记录集，返回内容为二维数组或 null；二维数组的结构是 aResult[X][Y]，X 为记录行，Y 为字段（例如 Select name, sex，那么 Y 就是 name,sex）；
  例如：aResult[0][‘name’]=’小王’; aResult[0][‘sex’]=’F’;
  aResult[1][‘name’]=’小张’; aResult[1][‘sex’]=’M’;
- $odb->db()->selectOne (null, 'SELECT COUNT(0) as CNT FROM tablename");
  返回 SQL 语句的第一条记录的第一个字段值或者为 null；（常用指令，性能优化）
- $odb->db()->selectFirstCol(null, 'SELECT name,sex FROM tablename');
  返回 SQL 语句结果集的第一列记录（性能优化），一维数组或 null；
  例如: aResult[0] =’小王’; aResult[1] =’小张;’
- $odb->db()->selectFirstRow(null, 'SELECT name,sex FROM tablename");
  返回 SQL 语句结果集的第一行记录（性能优化），一维数组或 null；
  例如：aResult[‘name’]=’小王’; aResult[‘sex’]=’F’;
- $odb->db()->selectRowCnt(null, 'FROM tablename WHERE ...");

  返回 SQL 条件的行数统计（性能优化）；$sSql 参数不需要存在 SELECT 头。

- $odb->db()->selectPage($iPage, $iPageSize, $bDBWR=null, $sSql=null);
  返回 SQL 语句的分页处理数据库，返回的结果集可参照 select()的二维数组结构；$iPage 是当前页码>=1，$iPageSize 是页大小；基于记录集总条数可使用 selectRowCnt()直接获取；

无返回记录集的操作：

- $odb->db()->exec($sSql);

可以执行 INSERT | DELETE | UPDATE|CREATE 等无返回记录集的 SQL 语句操作，执行后返回影响的记录条数。

2.2.6 数据库操作（中级：链式 SQL 的 CURD 操作）

如何防止 SQL 注入攻击：

在生成 SQL 语句的时候，如果不对外部传入的值做处理，直接拼接字符串到 SQL 将会使程序存在重大安全隐患，因此在拼接 SQL 字符串事要对以下几种数据类型做响应处理：

- 所有外部传入数字 Int | double 在拼接时必须做 intval()或 doubleval()处理，防止传入恶意字符串。
- 所有外部传入字符串处理时必须使用 CDbCURD::S('.....')这个静态函数对字符串做安全转义处理（防止字符串内容能被直接执行的可能），并且此静态函数输出处理结果中已经为字符串带了前后单引号，因此可以直接赋值；例如：’WHERE name=’. CDbCURD::S('小王'); 等同于’WHERE name=\’小王\’’;

表前缀的使用：

当你的数据库的表在设计时候使用表前缀方式命名（建议使用这种设计方式）在拼接 SQL 语句的 FROM 段时，可使用$odb->db()->TabN($sTableName)这个成员函数，输出的内容会自动根据 dsn 配置文件中设定的表前缀拼接在$sTableName 头部；例如:dsn 中设定的表前缀是'SAC_'，那么执行 $odb->db()->TabN('USERINFO')函数输出的内容为'SAC_USERINFO'；

```
$sSql = 'SELECT INDT, CVID FROM '.$odb->db()->TabN('client_visit'). ' WHERE INDT > \'2012-11-01 00:00:00\' LIMIT 5';
$odb->db()->selectFirstCol(null, $sSql)
```

[Code 2.2.6.1 代码片段]

链式 SQL 操作（CURD 操作）

SQL 链式操作可简化 SQL 代码，利用 IDE 的代码提示急速提升你的工作效率，实际使用方式如下。

链式 SQL 操作根据 SQL 语句的关键字分成了若干个不同的子函数，每隔函数可以链式的进行编写。CURD 的主操作分为 select、insert、update、delete；辅助操作为：

fields()：对应的 SELECT 后的字段部分，或 INSERT INTO 的字段定义部分；

where()：对应的是 WHERE 后面的条件字符串部分，可支持模板替换功能；

from()：对应的是 FROM 表名，且会自动做表前缀添加处理；

left_join()：必须在 from()后使用，用于生成表关联的操作；

param()：在 UPDATE 操作时表示 SET 子集中的内容，或 INSERT 的单条插入的记录集；

order()：能生成 SELECT | UPDATE 中 ORDER BY 运算关键字的内容；

group()：能生成 SELECT 中的 GROUP BY 运算关键字的内容；

having()：能生成 SELECT 中的 HAVING 运算关键字的内容；

详细使用参数请 IDE 中的自动代码提示。

```
//SELECT 基本处理
_dbg
(
$odb->db()
->fields(array('TITLE', 'URL', 'UPDT'))
->from('fun_url')
->where('SACID={@id}', array('{@id}'=>' xxxxxxx'))
->order('UPDT ASC')
->limit(3)
->select()
);
```

[Code 2.2.6.2 SELECT 基本处理 代码片段]

在[Code 2.2.6.2]中，->fields()参数是个一维数组,它生成了如下 SQL 段: SELECT TITLE, URL, UPDT；

->from('fun_url')参数为表名，它生成如下 SQL 段：FROM sac_fun_url；

->where(..., ...)第一个参数为条件字符串可用模板替换符（{@id}），替换符可以自定义。第二个参数为一个数组，数组以 key=>val 形式表示，key 为模板替换符，val 是需要替换成的内容。使用模板替换符的好处是，系统会判断传入的值是字符串还是数字，对于字符串会自动进行安全转换并加入前后单引号，对于数字将会直接输出替换符；

->order('UPDT ASC')参数为排序字，它生成如下 SQL 段：ORDER BY UPDT SAC；

->limit(3)参数为取前三条记录，它生成如下 SQL 段：LIMIT 3；

->select()指令表示链式操作结束执行 SELECT 动作，SQL 返回二维数组结果集，最终生成的 SQL 语句如下：SELECT TITLE, URL, UPDT FROM sac_fun_url WHERE SACID='xxxxxxx' ORDER BY UPDT SAC LIMIT 3；

```
//SELECT 表关联处理
_dbg
(
$odb->db()
->fields(array('TITLE', 'URL', 'UPDT'))
->from('fun_url', 'A')->left_join('A', 'fun', 'B', array('ariid'))
->where('SACID={@id}', array('{@id}'=>'xxxx'))
->select()
);
```

[Code 2.2.6.3 SELECT 表关联处理]

在[Code 2.2.6.3]片段中，->from('fun_url', 'A')的第二个参数是设定左关联表别名为 A，它生成如下 SQL 段：FROM sac_fun_url　A；

->left_join('A', 'fun', 'B', array('ariid'))的第二个参数 fun 是需要与别名 A 做左连接的关联表。第三个参数是 fun 的别名 B。第四个参数是一维数组，当只有一项时，表示同字段名的关联 A. ariid=B. ariid，当有数据有两项时（如 left_join('A', 'fun', 'B', array('ariid', 'kpid'))），表示 A. ariid=B. kpid 关联。最终生成的 SQL 语句如下：SELECT TITLE, URL, UPDT FROM sac_fun_url A LEFT JOIN fun B ON(A. ariid=B. ariid)　WHERE SACID=’xxxxxxx’

其余 SELECT 的链式操作指令类似，可参考上面的方式。

```
//insert 单条插入处理
_dbg
(
$odb->db()
->from('test')
->param(array('val_text'=>CDbCURD::S('insert():'.date('Y-m-d H:i:s'))))
->insert()
);
```

[Code 2.2.6.4 insert 单条插入处理]

[Code 2.2.6.4]是一个链式操作的 INSERT 插入操作，->param()的参数是一个 key=>val 结构数组，可以有多项。key 是字段名，val 是字段对应的值。对于 val 的值传入时需要注意，字符串必须用 CDbCURD::S()静态函数对其进行安全转换，否则有安全隐患。->insert()后及执行插入操作，它会生成如下 SQL 段: INSERT INFO sac_test (val_test)VALUES(‘insert()2012-01-01 12:00:00’);

```
//insertMulti 批量插入处理
$aData = array();
$aData[] = array(CDbCURD::S('abcd'));
$aData[] = array(CDbCURD::S('bbbb'));
$aData[] = array(CDbCURD::S('WXYZ'));
_dbg
(
$odb->db()
->from('test')
->fields(array('val_text'))
->insertMulti($aData)
);
```

[Code 2.2.6.5 insert 批量插入处理]

[Code 2.2.6.5]是批量 INSERT 操作的处理，->fields(array('val_text'))参数为一维数组表示要插入的字段名称；->insertMulti($aData)执行批量插入指令，其参数类型是二维数组，第一维是数据行，第二维是与 fields()中对应项的字段内容（即: fields()数组中有多少字段项，$aData 中的第二维就需要有多少与之对应的内容项）。最终它会生成如下 SQL 段:INSERT INTO(val_text)VALUES('abcd', 'bbbb', 'WXYZ')

```
//delete 处理
```

```
_dbg
(
$odb->db() ->from('test') ->where('id_num > {@id}', array('{@id}'=>6)) ->delete()
);
```

[Code 2.2.6.6 delete 处理]

[Code 2.2.6.6]是删除处理的代码片段，->where()表示匹配的条件，使用方式与之前介绍的 SELECT 中的方式一样；->delete()表示执行删除指令；最终它会生成如下 SQL 段：DELETE FROM sac_test WHERE id_num > 6；

```
//update 处理
  _dbg
(
$odb->db()
        ->from('test')
        ->param(array('val_text'=> CDbCURD::S('abcd')))
        ->where('id_num={@id}', array('{@id}'=>3))
        ->update()
);
```

[Code 2.2.6.7 update 处理]

[Code 2.2.6.7]是更新记录操作代码片段, -> param()表示更新的参数（即：SET 中的参数），类型为数组可存在多项，每项用 key=>val 形式表示；->where()为更新条件，使用方式见之前 SELECT 中的介绍；->update()为更新执行指令；最终它会生成如下 SQL 段：UPDATE sac_test SET val_text ='abcd' WHERE id_num > 3；

2.2.7 数据库操作（高级）

数据库表对象：

数据库表对象的做用是将数据库操作代码从业务逻辑层分离出来，以表为单位建立类文件（以表为单位为了能方便管理），将与该表相关的数据库业务处理抽象为类成员函数，以便于提升代码重用率与简化日后对系统的维护工作量。当对数据库内字段有修改时，不用在每个调用的业务逻辑中修改 SQL 语句，只需要在封装的表对象函数中，集中的将 SQL 语句做统一修改，就能完成维护的操作。

数据库表对象类文件存放的路径是[\SPFW\workgroup\db\table_object\]类文件名称必须大写，文件名与类名必须相同，类名必须是对应的数据库表名，并且需要继承 CdbTable 类。

优化提示：在表对象类中进行链式 SQL 的编写时，如果是对当前表操作，可以省略掉->from()这个方法，系统默认会给你置入表名。

接下来以一个实例来介绍数据库表对象类如何编写：

```
<?PHP
class TEST extends CDbTable
{
        public function addInfo()
        {
                return $this
                        ->param(array('val_text'=>'insert():'.date('Y-m-d H:i:s')))
                        ->insert();
        }

        public function getInfo()
        {
                return $this
                        ->fields(array('id_num', 'val_text'))
                        ->selectPage(2, 3);
        }
```

```
}
?>
```

[Code 2.2.7.1 表对象代码]

[Code 2.2.7.1]中 TEST 为数据库中的表名（注意要大写），它继承了 CdbTable 这个基类。TEST 表对象类中实现了两个函数方法：addInfo()与 getInfo()。例如 addInfo()函数，对数据库操作可以写成 $this-> param(....)->insert()这个形式，因为是表对象操作所以可以省略 from()；其他操作方式与之前的方法一样。

完成数据库表对象的编写后使用方法，请看下面的样例代码：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
require './SPFW/sea_php_init.php';
$odb = new CDB('CDbCfgSAC');

$oTable = $odb->tableObj(' TEST');
_dbg($oTable-> getInfo ());
$odb->db()->showHistory();
?>
```

[Code 2.2.7.2 表对象使用样例代码]

如[Code 2.2.7.2]所示，首先获得数据库操作对象$odb = new CDB('CDbCfgSAC')；然后获得表对象$oTable = $odb->tableObj('TEST')；此时如果 TEST 表对象文件不存在 tableObj()不会报错，它会直接返回一个没有自定义方法的表对象，你可以像[Code 2.2.7.1]中的函数一样直接操作，例如：$oTable->where('id>5')->selectOne();就能返回 TEST 表中 id>1 的记录统计值。

$oTable->getInfo()是[Code 2.2.7.1]中定义的成员函数，实现了分页输出 TEST 表中的数据；$odb->db()->showHistory()是一个调试语句，它会打印出之前执行过的 SQL 语句记录与执行时间。

因此使用表对象后在项目开发与维护方面能够极大的简化代码的复杂度与提升代码的重用率，让程序更容易阅读与管理，在团队协作时的缩短磨合时间。

分布式数据库优化，读写分离的处理：

当网站的访问量逐渐上升后，数据库的并发响应压力会上升很快，此时一般会使用数据库的同步复制功能将单纯的数据读操作分离到后备只读库上，而写操作依然保持在主库上操作，以便于将主要的数据读取操作转移到后备数据库服务器上，提升响应速度。

默认配置情况下系统的 RW(读写) 操作均在主库，可在[extend/db/lib/environment.cfg.php]中对默认读 RW 或 R 库进行设置。除此之外可以直接在 SELECT 的操作中，直接指定是访问主库还是只读库，详情参考前面介绍的 SELECT 方法中$bDBWR 参数。

SQL 运行日志记录

如果发现某些数据库语句处理导致严重影响站点的响应速度，可以打开[SPFW/extend/db/lib/environment.cfg.php]中的 write_log 项。启用日志后日志数据将会存储在[SPFW\log\core\extend\CDB]目录中，日志包含了两个信息，SQL 执行的时间与 SQL 串内容，可以根据这些日志信息来分析哪一条数据库指令导致性能下降，定对其进行优化。

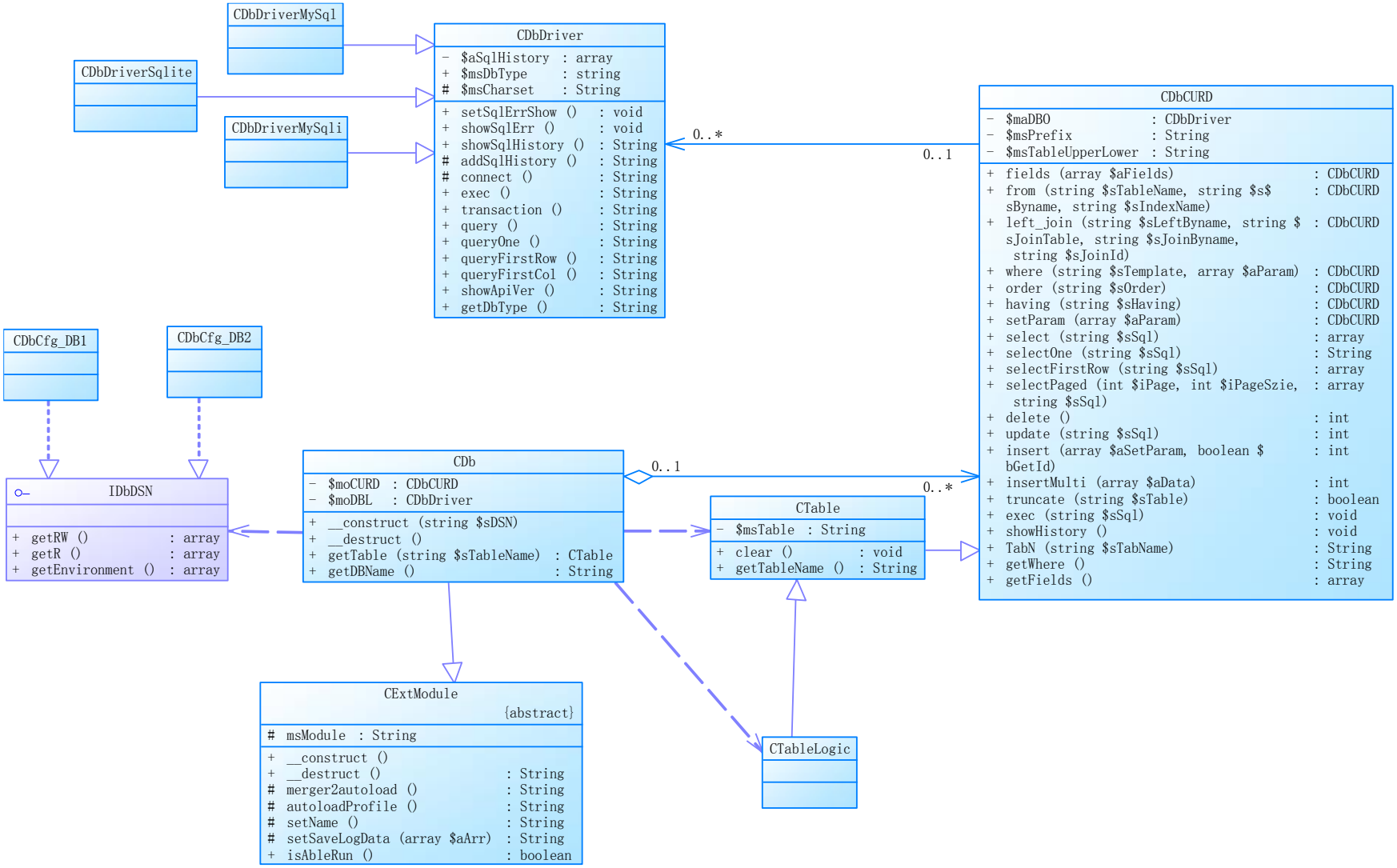
建议在生产系统的非必要情况下不要开启这个选项，否则将影响性能并可能生成大量日志信息。只有在检查系统性能时才打开这个选项。

下面为 sQL 日志文件的样例格式，其中第一句用于防止日志文件被下载。

```
<?php exit(0);?>
294.924 ms,       INSERT INTO test (val_text) VALUES ('insert():2013-05-15 08:54:46')
76.786 ms, INSERT INTO test (val_text) VALUES ('abcd'), ('bbbb'), ('WXYZ');
54.4651 ms,       INSERT INTO test (val_text) VALUES ('insert():2013-05-15 10:16:47')
```

[Code 2.2.7.3 SQL 系统日志样例代码]

2.2.8 DB 框架类继承关系结构图



[Image 2.2.8.1 DB 框架类继承图]

2.3、高速缓存模块介绍(cache)

本地文件缓存

建立在本地磁盘上的文件形式缓存，可将相对时间内保持不变的静态内容写入磁盘，在需要的时候直接读出，减少对数据库的 IO 操作，提高系统响应时间。例如页面缓存或从数据库中经过复杂处理后取得的大列表数据做缓存效果尤为明显。本地缓存文件存放在[SPFW/cache/filecache]目录下，你可以手动删除目录下所有的缓存文件，或者使用$oC->gc()函数来清除缓存，详情请看[Code 2.3.1.1]中的代码介绍。

Memcache 缓存（开源系统）

分布式内存缓存通过安装 memcache 服务组件，可将缓存独立到单一的服务器上，并且在缓存服务器上把数据以常驻内存的方式存储，能实现极高的并发响应性能（性能远超过文件式缓存）。

如何使用缓存模块

高速缓存执行模块存放位置在[SPFW\extend\cache\]目录下，使用前首先对[SPFW\extend\cache\config\environment.cfg.php]文件进行配置，设定当前使用哪个缓存类型（文件缓存或者 memecache）。如果使用 memcache 缓存还需对[SPFW\extend\cache\config\memcache.cfg.php]进行配置，设定缓存服务器的地址。

2.3.1 缓存模块使用方式

缓存模块共有五个主要操作方法分别为：get(): 获取缓存内容；set()设定一个缓存；del()删除一个缓存、increment()增量操作、decrement()减量操作；

使用样例：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
require './SPFW/sea_php_init.php';

$oC = new CCache();
_dbg($oC->isAbleRun()); //检查是否具备运行环境
$sKey = 'runtime';
$aRet = $oC->get($sKey); //获取名称为 Key 的缓存内容
if (is_null($aRet))
{
	$aRet = array
	(
		'name'=>'jerryli',
		'datetime'=>date('Y-m-d H:i:s'),
		'content'=>'共享锁缓存测试',
	);
	$oC->set($sKey, $aRet, 5); //保存名称为$sKey 的缓存，内容为$aRet，失效时间为 5 秒
}
_dbg($aRet, '数组缓存');

$iTmp = $oC->increment('visit', 2); //对 Key 为'visit'的缓存，增量 2
if (is_null($iTmp))
{   //缓存不存在，创建一个缓存
	$oC->set('visit', 1); //设定增量值初始值【注意必须为 int】
	$iTmp = 1;
}
$iTmp = $oC->decrement('visit'); //减量操作对 'visit' 缓存变量-1
_dbg($iTmp, '增量');
// $oC->del($sKey);   //删除名称为$sKey 的缓存
$oC->gc(); //回收所有缓存资源。
?>
```

[Code 2.3.1.1 缓存操作样例代码]

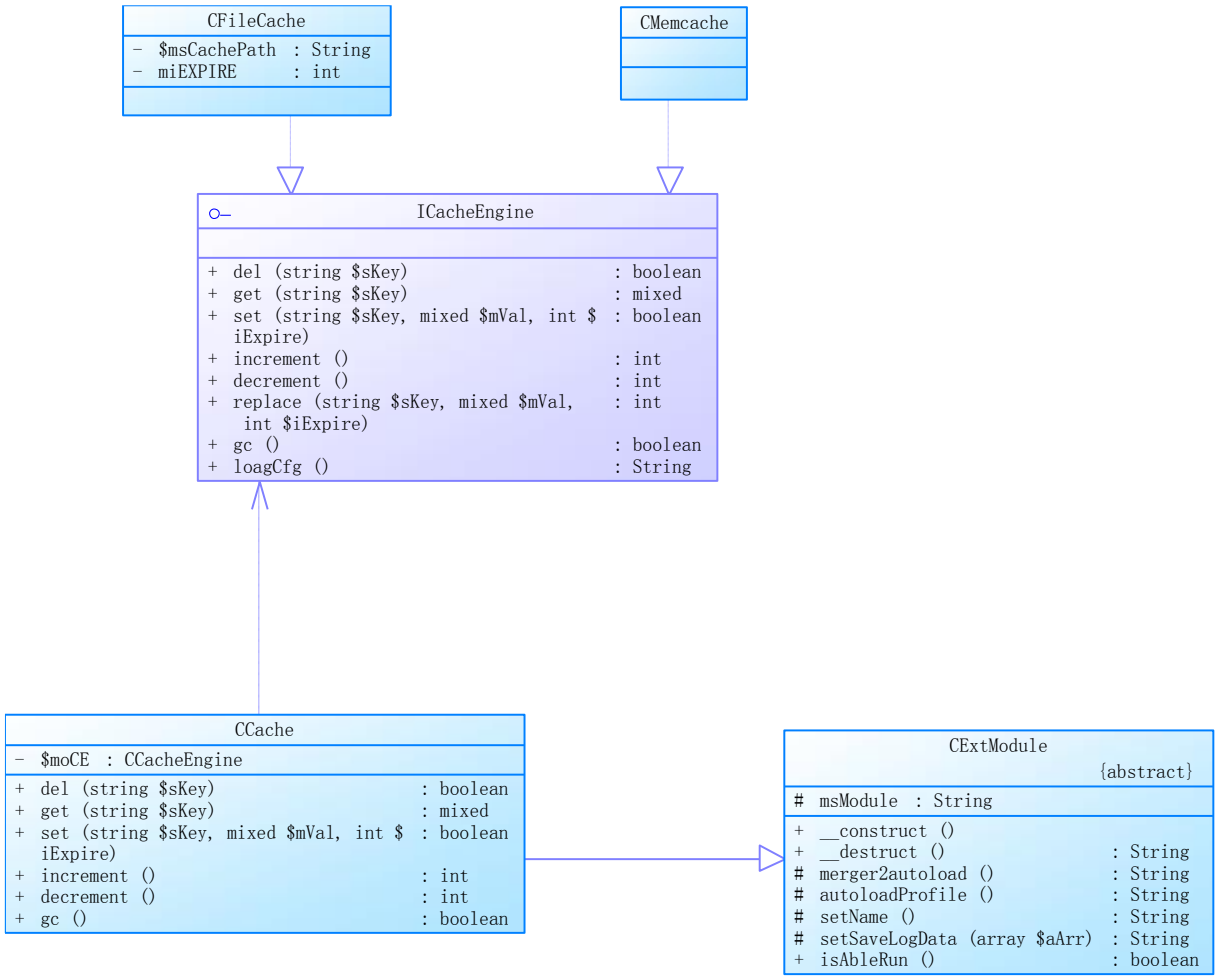
如[Code 2.3.1.1]所示，首次构造缓存对象$oC = new CCache();使用前建议用$oC->isAbleRun()检查当前的运行环境是否能运行缓存模块。

使用 get()方法检查某个缓存关键字是否存在，如果函数返回 null 表示未建过缓存或缓存超时，此时使用 set()函数设定缓存内容，下次再使用 get()函数就能根据关键字取到缓存内容（必须在缓存有效生命值之内）。

如果不需要某个缓存关键字可直接使用 $oC->del($sKey)这个方式删除缓存关键字。

对于文件缓存形式，为考虑系统效率它不会自动清除过期的缓存内容，因此随着使用时间的推移，缓存目录文件夹内容会增大，此时可以使用$oC->gc()释放所有缓存数据（或可在指定的时间清除缓存如凌晨三、四点的时候）。

2.3.2 缓存类继承关系结构图



[Image 2.3.2.1 缓存继承类关系图]

[Image 2.3.2.1]中，Ccache 为最终操作类，他继承 CextModule 扩展模块基类。

2.4、Web 站点设计框架引擎(website_engine)

2.4.1 框架特性

website_engine 框架遵循 MVC 设计模式，让开发者快速的开发 Web 应用，将主要的精力放在应用层而不是如何组织页面与性能问题。

框架采用单页面入口路由的访问形式，使用类似 Java 的包管理机制来管理业务逻辑程序，使得

网站的程度结构非常清晰，便于维护、排错、调试与团队的协作开发 。网站的页面模板部分使用了较为主流的 Smarty3 模板系统，如果你有自己的模板系统或者使用第三方模板系统也可以自行替换。

2.4.2 框架的目录结构

website_engine 层框架目录的内核位于[SPFW/extend/website_engine/]中，其中包含了 3 个目录 config、lib、runtime，首次使用需要在 config 目录中的文件进行配置。

目录结构：

SPFW/extend/db/ 数据库层内核目录
- config/ 环境配置文件
- lib/ 框架类库文件
- runtime/ 运行时文件(入口类)

SPFW/cache/website_engine 静态缓存目录

SPFW/workgroup/website_engine/logic 业务逻辑文件目录

SPFW/workgroup/website_engine/smarty_tpl 页面模板目录

2.4.3 先入为主：Hello World 的实现

接下来将演示如何创建一个 web 管理区站点页面的过程，在开始之前需要了解以下的几个基本的概念。

website_engine 框架把网站的最小单位定义为工作区，每个工作区是一个唯一的入口文件。各个工作区的 Session 相互隔离，因为工作区是以入口文件为应用边界，因此对工作区内的 session 使用方式进行了改造，请不要直接操作 php 自带的 session 函数。这个改造并不影响系统的性能，甚至还能无缝的将 session 存储替换成 memcache 方式，或者是将 session 存入数据库，从而实现不修改代码就能使网站适应分布式负载均衡的处理。

下面将通过 4 步来创建一个 hello world 功能的页面（似乎有点杀鸡用牛刀，不过通过这个步骤可以了解如何使用这个框架高效的创建与管理项目）

步骤 1：在根目录创建入口文件/user/index.php，内容如下:

```
<?php
header('Content-Type: text/html; charset=UTF-8');
define('SEA_PHP_ROOT', '/SeaPhp/');
require './SPFW/sea_php_init.php';
CExtendManage::Run(new CWebsiteEngine('user'));
?>
```

[Code 2.4.3.1 index.php 用户工作区入口文件]

其中 CExtendManage::Run(new CWebsiteEngine('user'))这行代码表示创建一个叫 user 的工作区，后面将会以这个工作区的名称(user)创建对应的业务逻辑区目录以及模板区的目录，目录名都必须为 user。

步骤 2：创建业务逻辑区目录；SPFW/workgroup/website_engine/logic 下创建 user 目录，并同时创建业务逻辑文件 hello_world.php（SPFW/workgroup/website_engine/logic/user/hello_world.php）。这里需要注意的是业务逻辑必须继承 CWebsiteModule 类，否则该逻辑无法被框架辨认。

```
<?php
class hello_world extends CWebsiteModule{
	/* (non-PHPdoc)
	 * @see CWebsiteModule::defaultFunc()
	*/
	public function getDefaultFunc()  {
		return 'login';
	}
	/**
	 * 登录
	 * @param string $sCtl
```

```
         * @return void
         */
        public function login($sCtl)  {
                echo "hello world";
        }
}
?>
```

[Code 2.4.3.2 hello_world.php 业务逻辑文件]

其中需要注意的几项：1、hello_world.php 与它内部的类名 hello_world 必须相同；2、必须实现 getDefaultFunc()函数，它的功能是告知当前类的默认启动方法为 login()，并同时实现 public function login($sCtl)这个函数的功能（此函数就为当前类中的某一个业务逻辑）；

步骤 3：配置工作区的默认启动类，找到 SPFW/extend/website_engine/config/entrance.cfg.php，在里面添加 user 工作区的默认业务逻辑类（即:【步骤 2】中创建的首页类名称）；

```
<?php
return array(
        'user' => 'hello_world',
);
?>
```

[Code 2.4.3.3 entrance.cfg.php 配置工作区的入口类]

步骤 4：创建入口访问安全检查的业务逻辑文件 SPFW/workgroup/website_engine/logic/user/WebLogicSecurityCheck.php。此文件将在后面[2.4.9 节]中重点讲到，其基本的功能是 user 用户区被访问时，系统对用户的权限进行检查，如果用户未授权将禁止访问或者跳转到某个登录页面中（如 session 不存在时要求用户登录）。下面的代码段中 package 为 hello_world 时将不对其做检查，默认通过。访问其他业务逻辑时会抛出错误提示。

```
<?php
final class WebLogicSecurityCheck extends CWebsiteSecurity{
        /* (non-PHPdoc)
          * @see CWebsiteSecurity::checkAccess()
          */
        public function checkAccess($sPkg, $sAct)     {
                //TODO:此处编写用户授权访问的业务逻辑
                if ($sPkg != 'hello_world') {//不对 hello_world 的任何做安全检查
                        echo '正在做用户访问权限检查:',
                        '<br/><strong>Pkg:</strong>', $sPkg,
                        '<br/><strong>Act:</strong>', $sAct,
                        '<br/>用户是否授权操作可修改这个类文件:'.__DIR__.'\\'. __CLASS__ .'.php',
                        '<br/>';
                }
        }
}
?>
```

[Code 2.4.3.4 WebLogicSecurityCheck.php]

到此一个 hellow world 的站点已经创建完成，可以访问 http://localhost/user/ 这个地址就能看到页面出现 hello world 的字样。(提示页面未找到？那需要配置 apache 默认启动文件中加入 index.php)；

同时也能这样访问 http://localhost/user/?hello_world-login 或者 http://localhost/user/?hello_world。为啥 http://localhost/user/ 这样访问时可以自动找到对应的逻辑页？因为你刚才配置了 SPFW/extend/website_engine/config/entrance.cfg.php 这个文件，同时在 hello_world.php 中你也指定了类的默认启动方法，所以它是智能的。

2.4.4 应用访问入口的业务路由原理

完成了 2.4.3 的构建首个工作区的体验后就需要转入正题了，website_engine 框架的入口访问路由是一个非常核心的概念，所有页面的访问都是由它开始的，也就是工作区的入口文件。

工作区的入口文件完成了 Web 内的业务逻辑定位，在外部的访问者看来这是很优雅的一种访问方式，访问者不会看到很长的目录结构，同时也不知道这个网站是 php 技术开发的，我们可以看到

整个访问过程都看不到 php 文件的扩展名，例如：http://localhost/user/ 或 http://localhost/user/?hello_world-login。

入口文件的路由格式如下：

http://localhost/user/	?hello_world	-	login	-	ctl	&a=..
工作区	class 或 package+'.'+class	分隔符	类方法	分隔符	控制参数	Get 参数
第一段	第二段		第三段		第四段	第五段

[Table 2.4.4.1 入口路由格式]

其中第一段是工作区，第二段是业务逻辑类名或者包路径+业务逻辑类名，第三段是方法名（即：业务逻辑类中的某个成员函数名称），第四段为控制参数（以 2.4.3 为例就是 public function login($sCtl) 函数的$sCtl 参数，这个值在路由逻辑处理是会直接抛入方法函数中），第五段如果还需要还可以加入 get 参数。

2.4.5 网站的业务逻辑规划

框架完全采用了面向对象的设计思路来看待网站的业务逻辑，因此把任何一个网站的业务逻辑功能都看成一个类中的一组操作方法。

举例说明：如果构建一个 crm 系统，在做系统分析时会规划出 N 个模块，如用户模块（包含注册，修改密码，找回密码等等），授权验证模块（包括登录验证，退出登录等等），报表模块（包含业务销售报表，库存报表，工资报表等），客服支持（包含内部工单处理，客户数据查询等等）。

按上面的这些功能开发的 CRM 系统，一个类模块未必能放下所有的方法，因此需要把一个业务模块分解成多个类，将多个类组成一个包，比如报表模块可以设计成 /report/sell、/report/wage、/report/stock，然后在对应的目录下编写对应处理的多个业务类，在类中编写对应的业务逻辑方法。这样就能使得项目的文件维护非常方便与直观。

按照/report/sell 为例，设在里面有 2 个业务逻辑类，product_rep.php 与 customer_rep.php。product_rep.php 中有一个 show_month_report()方法显示产品的月报表，对应的入口文件访问的路径为 http://localhost/crm/?report.sell.product_rep-show_month_report-list&year=2012&month=11。业务逻辑类文件的存放路径为 SPFW/workgroup/website_engine/logic/crm/report/sell/product_rep.php。接着在对应的类文件中编写相应的方法就行，整个网站的开发过程就按照这个思路来做。

2.4.6 页面模板 Smaty3 框架的调用

website_engine 中内嵌了 Smarty3 的模板引擎框架（Smarty3 模板的详细使用方式 http://www.smarty.net/docs/zh_CN/），通过$this->view 对象可直接使用。调用方式很简单就 2 个方法：

$this->view->O($sKey, $Data); 通过这个函数将变量送出到模板，有 N 个参数就调用 N 次。

$this->view->showPage($sTpl); 通过这个函数显示模板，$sTpl 是模板的名称（如: report_list.tpl）。注意 showPage()函数不会终止程序，需要自行在函数后面执行 return 防止后面的程序执行（或者将这函数的调用放在函数逻辑的最后一行）。

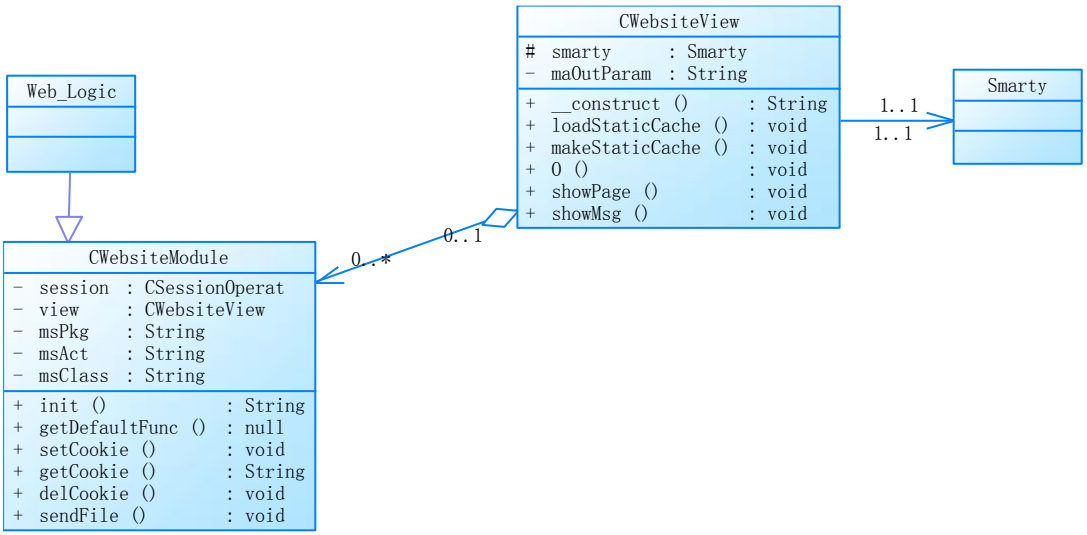
网页的静态模板文件存放在 SPFW/workgroup/website_engine/smarty_tpl/[工作区]/ 内。Smarty3 静态框架文件的编写请自己学习。

业务逻辑中调用模板示例代码如下：

```
<?php
class FinanceManage extends CWebsiteModule{
    /*此处省略 1000 字*/
    public function consumeMoneyList($sCtl)    {
        $sAct = '代理款流水帐';
        $this->view->O('aAgentInfo', ...); //代理扩展信息
        $this->view->O('RowTotal', ...); //总记录条数
        $this->view->O('Rows', ...); //记录集
        $this->view->O('Page', ...); //当前页
        $this->view->O('PageSize', 30); //页大小
```

```
            $this->publicTagOut($sAct);
            $this->view->showPage('FinanceManage_consumeMoneyList.tpl');
        }
}
?>
```

[Code 2.4.6.1 业务逻辑类中调用模板示例]



[Image 2.4.6.2 CWebsiteView 对象]

通过上面的类图可以看出 CWebsiteModule 中有 CWebsiteView 对象的一个实例 view，其中的 o()、showPage()、showMsg()3 个函数定义了对 smarty3 模板库的操作。

showMsg()函数定义了一个公共的信息提示页，当需要提示用户：例如密码错误，或者设置成功，设置失败之类的统一格式的提示页时直接调用这个函数，如果您要对这个提示页的样式与现实内容结构做调整，那么可以在工作区的模板目录中找到 showMsg.tpl 模板文件直接对其修改（路径如下 SPFW/workgroup/website_engine/smarty_tpl/[工作区]/showMsg.tpl）。下面是这个提示页的大致效果。

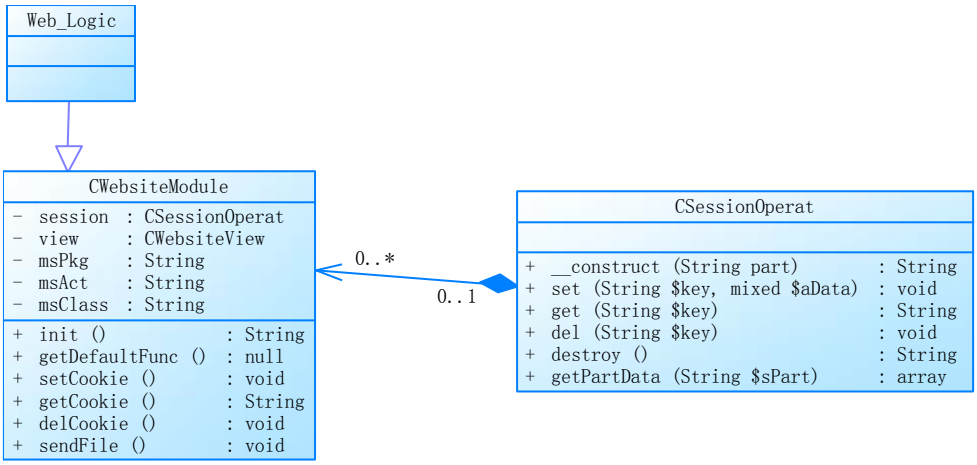
[Image 2.4.6.3 showMsg 的显示效果]

showMsg()函数的使用代码如下(详细参数请直接看函数的自动代码提示)：

```
$this->view->showMsg('succ', '新密码设定成功。', '密码设置', '?SysManage-changePwd');
```

2.4.7 工作区内的 Session 使用

由于 website_engine 的设计架构不是通过二级域的方式区分工作区（传统网站一般以二级域来区分工作区，比如 agent.abc.com、crm.abc.com、www.abc.com），而是通过入口文件来区分工作区，所以如果使用 php 自带的 session 函数会存在互相覆盖的情况，因此 website_engine 框架对 session 进行了重构，并封装了 CSessionOperat 类。通过这个类可以让 session 的操作变得异常简单。



[Image 2.4.7.1 CSessionOperat 类]

通过上面的 CSessionOperat 类图可以发现，session 类的操作提供了 set()、get()、del()、deatory()这 4 个操作函数。存储 session 时可以使用 set($key, $aData)函数，$aData 可以是 Array，字符串，数字。注意：尽量不要存储 object 到 session，这会增加服务器的内存开销。

下面这个 getAllList.php 的业务逻辑演示了 session 操作的示例，deluser()函数为默认启动函数：

```
<?php
class getAllList extends CWebsiteModule{
	function __construct() {
		echo __CLASS__ .'::'. __FUNCTION__ ."():i'm here:";
	}

	public function getDefaultFunc(){
		return 'deluser';
	}

	public function deluser($sCtl){
		if (is_null($sBuf = $this->session->get('user.list')))
			$this->session->set('user.list', 'ttete'.date('Y-m-d H:i:s'));
		else
			echo "<br/>Session:". $this->session->get('user.list');
	}
}
?>
```

[Image 2.4.7.2 getAllList.php 演示 Session 操作]

deluser()函数演示了首先获取 key 为'user.list'的 session 内容，如果不存在则设置一个新的 session 值，如果存在 key 则取出 Session 值

2.4.8 页面的高速缓存

如果网站的访问压力很大，解决响应速度最直接的方法是使用静态缓存，website_engine 框架在页面逻辑级别提供了静态缓存的功能（如果你需要使用更细的代码数据级别的缓存可参考 2.3 节的介绍）。

静态页面缓存未必适用于所有场景。例如大鼠局查询就非常适合使用，或者产品介绍页等。在页面首次被访问的时候会根据您给定的关键字规则将当前的整页输出结果缓存到文件，当用户下次访问的时候，命中了你存储的关键字规则时立即将之前缓存的页面读出推送出去（这个效率较高一般是个位数的毫秒级别）。

高速缓存的函数在 view 对象内直接使用，共有 2 个函数 loadStaticCache()与 makeStaticCache()。使用逻辑如下：

1、 首先判断是否存在缓存 loadStaticCache($sKey,$iLifeTime)，如果返回 null 表示未命中缓存，其中$key 表示关键字（一般为用户 ID 加上一系列搜索条件的字符），$iLifeTime 表示这个

缓存的有效时间（虽然通过关键字命中了缓存但是此缓存超过了$iLifeTime 的限定时间将会丢弃这个缓存）。

2、 当缓存未命中时执行正常的后续业务逻辑，并最终输出结果集后，调用 makeStaticCache($sCtl);函数 $sCtl 参数是自定义的缓存关键字，此函数执行后会将页面的输出内容缓存到本地文件中，等待下次的调取。

页面的缓存文件会存储在 SPFW\cache\website_engine\static_cache\[工作区] 这个路径下，因此需要保证 SPFW\cache 目录及下面的子目录都要有可读写的权限，否则会无法写入缓存文件。

页面缓存的示例代码如下：

```
<?php
class getAllList extends CWebsiteModule{
	function __construct() {
		echo __CLASS__ .'::'. __FUNCTION__ ."():i'm here:";
	}

	public function getDefaultFunc(){
		return 'deluser';
	}

	public function deluser($sCtl){
		$this->view->loadStaticCache($sCtl, 5); //读取缓存，且 5 秒内的缓存有效
		if (is_null($sBuf = $this->session->get('user.list')))
			$this->session->set('user.list', 'ttete'.date('Y-m-d H:i:s'));
		else
			echo "<br/>Session:". $this->session->get('user.list');

		echo '<br/>e:'. G('e');
		echo '<br/>b:'. G('b');
		for ($i=0; $i<10000; $i++)
			echo date('Y-m-d H:i:s');
		echo "<br/>function:". __FUNCTION__.'()';
		$this->view->O('user', 'jerryli');
		$this->view->showPage('child1.tpl');
		$this->view->makeStaticCache($sCtl);
	}
}
?>
```

[Code 2.4.8.1 页面高速缓存示例]

2.4.9 用户的授权访问业务逻辑

网站的用户授权访问业务逻辑一般常用的是 3 种策略，但核心都是采用 Session 检查对应位置否存在的方式：

1、 发现用户未登录（对应的 session 项不存在）则将用户访问页重定向到登录页面，强制用户登录，这个是最常用的简单登录验证方式。

2、 当用户未登录时只能受限访问页面中的某些功能，比如购物网站或者论坛常常采用这种方式。

3、 用户登录后，根据所属的权限控制字，判断可以对那些业务逻辑进行访问，哪些业务逻辑限制访问。一般的 CRM 系统或者内部信息管理系统常采用这种方式。

授权访问逻辑，在 website_engine 框架中的工作区目录内通过 WebLogicSecurityCheck.php 文件进行控制。

上述第二种授权登录方式，我们可以简单的跳过 WebLogicSecurityCheck.php 授权访问逻辑，在框架层面不做授权访问逻辑的处理。跳过 WebLogicSecurityCheck.php 授权访问逻辑的代码如下：

```
<?php
final class WebLogicSecurityCheck extends CWebsiteSecurity
{
```

```
	public function checkAccess($sPkg, $sAct)
	{
	}
}
?>
```

[Code 2.4.9.1 不做授权验证处理的 WebLogicSecurityCheck]

在[Code 2.4.9.1]的 checkAccess()函数中不写任何逻辑它就默认不处理授权验证的逻辑。

WebLogicSecurityCheck 业务逻辑当用户访问上来时，会执行 checkAccess($sPkg, $sAct)函数，并将[Table 2.4.4.1 入口路由格式]中的第二段传入$sPkg 参数，第三段传入$sAct 参数。然后您根据这两个参数来判断用户能够访问哪些业务逻辑。同时也能结合 session 的值判断用户是否登录过。对于存在 session 的用户可在这儿继续判断，当前的业务逻辑$sPkg 与 $sAct 是否允许他访问，如不允许访问就直接给出提示页。

如此您就不需要在业务逻辑的实现中考虑用户授权访问的问题，所有授权访问的业务逻辑都在入口的时候在 WebLogicSecurityCheck.php 中一并处理掉。

下面的代码就演示了类似的处理，以供参考。

```
<?php
final class WebLogicSecurityCheck extends CWebsiteSecurity{
	/**
	 * 公开的无需登录验证的逻辑方法
	 * @var array
	 */
	static $maPublicList = array('LoginOpt', 'ajax.LoginOrReg');
	/* (non-PHPdoc)
	 * @see CWebsiteSecurity::checkAccess()
	 */
	public function checkAccess($sPkg, $sAct){
		if (!in_array($sPkg, self::$maPublicList)){
			if(is_null($this->session->get('agent_login')))
				redirect('?LoginOpt-login');//会话丢失,要求重新登录
		}
	}
}
?>
```

[Code 2.4.9.2 相对复杂的授权验证处理 WebLogicSecurityCheck]

在[Code 2.4.9.2]中开头处定义了$maPublicList 数组，数组存放的是哪些 class 的访问不做授权验证处理。在 checkAccess()函数中首先判断当前访问者的访问$sPkg 是存在于$maPublicList 数组中，如果不存在时表示需要对用户进行授权验证，此时会通过$this->session->get()检查对应的 session 数据位是否存在，不存在强制将页面重定向到登录界面，让用户登录后再访问。

2.4.10 在业务逻辑内使用数据库模块

这里所指的是在 website_engine 框架下的网站业务逻辑类文件中，如何使用 2.2 节中介绍的数据库模块，其步骤非常简单。

由于 SeaPHP 的 extend 中每个扩展模块都是完全独立的（高内聚低耦合），因此在 website_engine 框架中没有直接整合数据库模块，使用时需要在入口文件中实例化一个数据库操作对象（[Code 2.2.5.1]中展示了如何实例化一个数据库操作对象），然后在业务逻辑类中 global 这个数据库全局操作对象进行处理。

下面将演示一下如何在 website_engine 的工作区中使用数据库层的操作示例。

首先在 website_engine 工作区的入口文件中加入数据库对象的实例化代码，下面将以 agent 工作区为例（/agent/index.php）：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
require '../SPFW/sea_php_init.php';
```

```
$mDB_WP = new CDB('CDbCfgLocalWeiPlug'); //实例化数据库操作对象
CExtendManage::Run(new CWebsiteEngine('agent')); //实例化 agent 工作区
?>
```

[Code 2.4.10.1 agent 工作区入口文件]

接着在需要使用数据库操作的页面逻辑中的构造函数__construct()内引入$mDB_WP 全局变量，构建表对象后对数据表进行一系列的操作，示例代码如下：

```
<?php
class LoginOpt extends CWebsiteModule
{
	/**
	 * 代理表
	 * @var AGENT_ACCOUNT
	 */
	private $mdbAgentAccount = null;

	/**
	 * 构造函数
	 */
	function __construct()
	{
		/*数据库对象*/
		global $mDB_WP; //获取数据库操作引用
		$this->mdbAgentAccount = $mDB_WP->tableObj('AGENT_ACCOUNT');
	}

	/* (non-PHPdoc)
	 * @see CWebsiteModule::defaultFunc()
	*/
	public function getDefaultFunc()
	{
		return 'login';
	}

	/**
	 * 代理登录界面
	 * @param string $sCtl
	 * @return void
	 */
	public function login($sCtl)
	{
		$sAct = '代理登录';
		$sLogname = P('log');
		$sPwd = P('pwd');
		if (empty($sLogname) || empty($sPwd)) //使用静态缓存
			$this->view->loadStaticCache();

		if (in_array($sCtl, array('do')))
		{
			$sAGID= $this->mdbAgentAccount->loginCheck($sLogname, $sPwd);
			if (!is_null($sAGID))
			{	//登录成功
				$aBase = $this->mdbAgentAccount->getInfo($sAGID); //取出代理信息
				$this->session->set('agent_login', 'ok');
				$this->session->set('agent_logname', $sLogname); //登录帐号
				$this->session->set('agent_name', $aBase['AGNAME']); //代理名称
				$this->session->set('agent_agid', $sAGID); //代理 ID
				unset($aRet);
				redirect('?SysManage-main'); //跳转到代理区
			}
		}
		$this->view->O('sAct', $sAct);
		$this->view->showPage('LoginOpt_login.tpl');
		$this->view->makeStaticCache(__CLASS__.__FUNCTION__); //建立静态缓存
```

```
    }
}
?>
```

[Code 2.4.10.2 页面业务逻辑页]

[Code 2.4.10.2]是一个系统的代理登录验证页面，__construct()中引入了入口文件中的$mDB_WP 数据库操作对象，并创建了一个表对象引用 $this->mdbAgentAccount = $mDB_WP->tableObj('AGENT_ACCOUNT'); （表对象引用部分的知识可看 2.2.7 章节）。接着在 function login($sCtl)方法中实现了用户登录的业务逻辑，使用内核库函数 P()操作获取用户 POST 过来的用户名与密码参数，并判断当$sCtl 为”do”时(即：http://localhost/agent/? LoginOpt-login-do)，检查用户是否有效$this->mdbAgentAccount->loginCheck();如果有效则建立 Session 会话后通过 redirect('?SysManage-main');函数重定向到管理区首页。如果$sCtl 不为 do 时直接显示用户登录页面。

通过读上述代码可以很容易理解整个业务逻辑，所有数据库的操作都已经封装到了 AGENT_ACCOUNT 这个表对象内（即在页面逻辑中不会出现数据库层操作的细节代码），因此从业务逻辑上来看整个程序代码非常简炼，层次清晰。

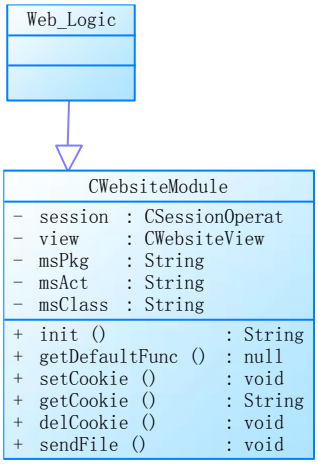
2.4.11 工作区内的 cookie 使用

业务逻辑中，可直接使用的 cookie 函数总共有 3 个，分别是 setCookie()、getCookie()、delCookie()。Cookie 通过 Key 的方式进行存储。在前台页面 UI 中使用 js 的读取（document.cookie），Cooike 串的结构是 key=val;…这样的存储格式，那么通过 setCookie()函数输出的 Cooike 的 key 命名规则是”[工作区]_[key]”。例如：后台设置的 setCookie()的 key 是 uname，工作区为 agent，那么前台 UI 通过 js 取到的 document.cookie 为 agent_uname=xxxxxx;这样的值。

为了便于前台使用方便在 SeaPHP 的 JS 库中提供了一个获取 cooike 的 jQuery 类文件位置在 /js/jQuery/ext/util/jQuery.cookies.js 详细的使用方式可直接看文件的注释。

2.4.12 业务逻辑基类 CWebsiteModule 文件下载函数

在某些时候，存在需要将输出结果变成一个文本文件或者.csv 文件让用户下载，而不是直接在网页中显示这个文本文件的内容的需求，这个问题可以用 CWebsiteModule 基类中的 sendFile()函数来解决，他能让浏览器弹出保存当前页面的窗口，而不是显示当前页面输出的内容。类图如下



[Image 2.4.12.1 CWebsiteModule 基类结构图]

2.4.13 website_engine 的公共静态函数

在框架中封装了最常用的几组方法：1、G()获取用户提交的 Get 参数;、2、P()获取用户提交的 POST 参数；3、PG()获取用户提交的 Post 与 Get 参数；4、redirect()浏览器页面重定向。

这些函数可直接在业务逻辑内使用，因为是静态函数执行效率很高。

2.4.14 文件上传模块

文件上传模块是最常用的一个功能，但是这并不属于框架内的功能，如果你想自己编写文件上传组件，可完全不用理会 uploadfile.php、upload_test.htm 两个文件，可完全使用自己的业务流程来处理。

SeaPHP 提供了一个 swf 的解决方案，同时该文件上传模块还支持将文件传送到阿里 OSS 的功能（阿里云 OSS 将在[2.6 节]中讲到）。

文件上传模块的后台部分的程序在根目录下的/uploadfile.php。如果要开启阿里云 OSS 转存，需要找到 36 行的下面这个代码：

```
    /**
     * 是否上传到阿里 OSS 云存储空间
     * @var bool
     */
    static $mbToAliOSS = false;
```

[Code 2.4.14.1 uploadfile.php 设置 alioss 上传]

将[Code 2.4.14.1]中的$mbToAliOSS 设置为 true，同时需要配置好 alioss 扩展模块的账户信息，同时要确保根目录下有 FileStore 目录，并且有读写的权限，这样就完成了后台的配置。

前台 html 部分的上传按钮可参考/ upload_test.htm 这个文件。你可将文件中的 js 代码以及包含文件复制后放到你需要上传按钮的页面中，js 代码中有几个配置参数较为重要，下面对其重点讲解，首先看代码：

```
var img_id_upload=new Array();//初始化数组，存储已经上传的图片名
var i=0;//初始化数组下标
$(function(){
    $('#file_upload').uploadify(  {
        'auto'        : true,//自动上传
        'removeTimeout' : 1,//文件队列上传完成 1 秒后删除
        'swf'         : './css/uploadify.swf',
        'uploader' : 'uploadfile.php',
        'method'    : 'post',//方法，服务端可以用$_POST 数组获取数据
        'buttonText' : '选择文件',//设置按钮文本
        'multi'      : false,//允许同时上传多张图片
        'uploadLimit' : 10,//一次最多只允许上传 10 张图片
        'fileTypeDesc' : 'Image Files',//只允许上传图像
        'fileTypeExts' : '*.gif; *.jpg; *.png; *.mp3',//限制允许上传的图片后缀
        'fileSizeLimit' : 2*1024,// 文件上传限制(2M:2*1024KB)，(可读取 PHP.ini 的配置，
ini_get('upload_max_filesize'))
        'formData': {'package':'lijian@abc_com.imgtext', 'rename':'test12345'},
        'onUploadSuccess' : function(file, data, response){
            //每次成功上传后执行的回调函数，从服务端返回数据到前端
            img_id_upload[i]=data;
            i++;
            alert(data);
        },
        'onQueueComplete' : function(queueData){
            //上传队列全部完成后执行的回调函数
        }
        // Put your options here
    });
});
```

上传组件使用的是开源的 http://www.uploadify.com/，详细的配置信息可直接网上搜索。本处的引用需要关注这几个参数：

'uploader' : 'uploadfile.php'，设置上传文件的后台处理程序；

'fileTypeExts' : '*.gif; *.jpg; *.png; *.mp3'，限制上传文件的类型；

'formData'，这个非常重要，'package'表示存储的位置，在根目录下以 FileStore 为基础的目录名（如：lijian@abc_com.imgtext，指的是 FileStore/lijian@abc_com/imgtext/这个路径），如果存在 rename

参数将强制对文件改名，不提供时保持原文件名。这几个参数会传给后台的 uploadfile.php 程序，详细的处理流程可直接查看 uploadfile.php 的源码。

'onUploadSuccess'，为上传完成后的回调函数，通知用户上传成功，并做相应的 js 处理。

2.5、mail 邮件模块

邮件模块是 web 项目中最常用的模块，SPFW\extend\mail 这个邮件模块可以让邮件的发送调用异常简单，并且支持 html，Text 形式的邮件，可以带附件，支持模板发送，只需要一次配置就能在任何地方使用。

2.5.1 配置邮件模块

邮件的配置文件路径是 SPFW/extend/mail/config/environment.cfg.php，配置信息如下

```
<?php
/**
 * Mail 模块的配置文件
 *
 * @var array
 * */
return array
(
'mail_host' => 'mail.abc.net', //smtp 发件服务器地址
'account_name'   => 'do-not-reply@abc.net', //发件人账户的用户名(smtp)
'account_pwd'    => 'lijian12', //发件人账户的密码
'from_address'   => 'do-not-reply@abc.net', //发件人邮件地址(一般与发件人账户名相同)
'from_name'      => '系统(请勿回复)', //发件人名字
'monitor_addr'   => 'lijian@abc.mobi', //默认监控地址(抄送地址)
//邮件模板目录(目录下的 text:文本邮件模板, html:网页邮件模板, signature:签名模板)
'template'   => 'workgroup.mail.template',
);
?>
```

[Code 2.5.1.1 mail 配置文件]

按照注释的要求配置好后就可以使用模块进行邮件的发送，需要注意的地方是'template'指向的目录中存放着邮件模板目录，此目录中有 3 个子目录分别是 html，text，signature。html 中存放着超文本邮件模板，text 中存放着纯文本邮件模板，signature 中存放的是邮件签名模板。

使用邮件模块发送邮件的调用方式：

```
<?php
header('Content-Type: text/html; charset=UTF-8');
define('SEA_PHP_ROOT', '/SeaPhp/');
require './SPFW/sea_php_init.php';
$oM = new CMail(); //邮件对象

$sRep = array('{#AgentName#}'=>'小王', '{#webhost#}'=>'web.weiplug.com', '{#agentid#}'=>'1231231223',
            '{#token#}'=>'123weasdf34e234sadf', '{#logname#}'=>'agent0000', '{#pwd#}'=>'123123');
$sTemp = $oM->getTemplateText('create_agent', $sRep, 'DoNotReply');
if(!$oM->Send(array('hzjerry@abc.com'), 'test mail', $sTemp))
    echo "邮件发送失败.";
else
    echo "邮件发送成功";
?>
```

[Code 2.5.1.1 mail 配置文件]

[Code2.5.1.1]中演示了使用邮件模块发送邮件的样例，其中$oM = new CMail();创建邮件对象，然后调用$oM->getTemplateText()读取一个文件模板并将模板中的替换符传入（如果读取超文本模板，则使用 getTemplateHtml()函数），最后调用$oM->Send()函数发送邮件。邮件签名模板只有在发送纯文本邮件是才有用参见 getTemplateText()的第三个参数。

邮件模板中可以根据自己的喜好设定占位符，代码[Code2.5.1.1]中$sRep 就定义了一个模板替换

数组，Key 是占位符，val 是待替换的值。下面是[Code 2.5.1.1]对应的 create_agent.txt 模板的示例代码：

```
{#AgentName#},您好！
    您的登录帐号: {#logname#}
    管理密码为: {#pwd#}
    agent 平台地址:http://{#webhost#}/agent/
```

[Code 2.5.1.2 create_agent.txt 模板示例]

2.6、阿里云 OSS 存储模块

此模块是使用了阿里云的 OSS 服务，可以通过简单的接口完成将数据文件上传到 OSS 服务器。

Alioss 的 SDK 请参见: http://aliyun_portal_storage.oss.aliyuncs.com/oss_api/oss_phphtml/index.html

使用模块前需要对 OSS 帐号信息配置（文件在 SPFW/extend/alioss/config/environment.cfg.php）。完成配置后可直接使用$oss = new CAliOSS(); 语句来创建 oss 对象。OSS 模块提供了如下几个操作方法；

put($sKey, $resFile, $iFileSize)函数：上传文件资源到 OSS。

pull($sKey, $sSaveFile)函数：拉取 OSS 文件到本地。

del($sKey)函数：删除 OSS 上对应的对象。

deleteAll($Prefix)函数：需要批量删除的对象键前缀。

需要注意的是 AliOSS V2 版本需要 PHP 版本最低要求是 5.3.2，否则无法使用。

下面演示如何上传一个文件，这个是相对复杂的一个操作，了解这个操作后其他几个函数就很好理解。

```
$resFile = fopen($sLocalPath . $sFileName .'.'. $sFileExt, 'r');
$iFileSize = filesize($sLocalPath . $sFileName .'.'. $sFileExt);
$oss = new CAliOSS(); //oss 操作对象
$sAliOSSPath = $oss->put($sKey, $resFile, $iFileSize); //上传文件
unset($oss);//释放对象
```

[Code 2.6.1 oss 文件上传实例]

[Code 2.6.1]其中第一行代码用来打开一个本地文件并获得一个文件句柄$resFile 然后通过$oss->put($sKey, $resFile, $iFileSize)将文件上传到 oss。

三、架构内核函数库的使用

3.1.1 Debug 函数

开发中使用最多最频繁的函数就是调试函数，用于打印运行环境中的信息到页面。系统提供了两个调试函数：_dbg($obj, $sTitle=null)；_dbge($obj)；

- _dbg($obj, $sTitle=null) 第一个参数为任意需要打印的对象，第二个参数为显示时的标题，用于区分是哪条调试语句打印出来的内容。函数在打印调试信息后不程序还会保持继续运行。
- _dbge($obj)；参数$obj 为任意需要打印的对象，打印结束后立即终止后面程序的运行。

```
//debug
_dbg('testing.....', 'memory'); //打印字符串
_dbg(1, 'memory'); //打印数字
_dbg(array('guest', 'todo', 'where'), 'memory'); //打印数组
_dbg(new CDB('CDbCfgLocalTest'), 'memory'); //打印类对象
_dbge('game over', 'run time'); //打印后立即结束程序的执行
echo 'run here..';
```

[Code 3.1.1.1 Debug 函数代码]

如[Code 3.1.1.1]代码执行后，最后一行’run here..’字符是不会被打印出来的，因为它上面有_dbge()

函数，当_dbge()输出’game over’以后程序就立即终止。

3.1.2 系统全局变量管理类

常用的全局参数可以通过全局静态类来直接获取，CENV 类有 4 个静态方法：

CENV::getCharset()获取系统字符集；

CENV::getRuntime()获取系统执行时间，这个时间指从接到用户对页面访问的时间算起的时间，毫秒；

CENV::getIP()获取访问者的 IP；

CENV:: getHost()获取访问者的主机头；

四、扩展框架或全局公共模块的开发

SeaPHP 具有很强的扩展性，您可以按照基本的规范开发一个自己的模块，将他加入 extend 中。在团队协作中可方便的供别人调用，同时也能发布到开源系统中，任何人都可以下载并加入到他系统的 extend 中共享您的代码。

4.1.1extend 的开发规范

SPFW/extend 目录下的所有扩展模块的入口程序都必须继承 CExtModule 类，如果开发的是框架(例如 website_engine)还必须实现 IExtFramework 接口中的方法，扩展模块内的目录结构请参照如下建议 config、lib、runtime。config 放置配置文件，lib 放置依赖库文件，runtime 放置入口启动类与静态函数库。如果是框架则用户的业务逻辑类实例需要存放在 SPFW/workgroup/[模块名]下面，以便于将框架的内核与应用隔离，方便日后的维护。

需要文件写入的操作时，以下两个目录有写入权限但不要授予可执行的权限：SPFW/log 用于写入日志文件，SPFW/cache 用于写入缓存文件，【注意】在 SPFW 目录内除了这两个目录以外其他目录都不要有写入权限，以保证内核的安全。如果有需要下载文件的需求，可将下载文件的目录放置到 SPFW 目录以外的地方。

引入第三方扩展模块或框架时，还需要在 SPFW/core/config/extend_framework.cfg.php 文件中加入运行时的入口文件的配置记录，此后就可以在任何地方使用该扩展模块了。

4.1.2 扩展模块的开发

下面将以前面的 2.3 章节为基础讲解如何开发扩展模块。首先从类图结构入手较直观，请找到[Image 2.3.2.1]这个类图（此图在 2.3.2 章节内）。

图中右下角是 CExtModule 基类，需要开发的扩展模块为左下角的 CCache 高速缓存类。CCache 继承 CExtModule 这个抽象类后，需要实现里面所有的抽象方法函数，接下来介绍每个函数的功能。

__construct()构造函数：此为固定格式，首先调用父类的构造函数，然后载入自己的配置文件$this->loadCfg()（例如调试信息标志位，账户信息的载入等等），配置文件建议存放在 SPFW/core/config/下，名称建议为 environment.cfg.php，示例代码：

```
function __construct()
{
	parent::__construct();
	$this->loadCfg();
}
```

[Code 4.1.2.1 扩展类入口程序之__construct()]

__destruct()析构函数：比为固定格式，最后一行执行父类的析构函数 parent::__destruct();

autoloadProfile()抽象函数：这个函数内需要实现扩展类的自动加载类文件的列表，这样在模块被加载时会将需要用到的私有依赖类文件注入到内核的自动类加载数组中。函数的实现参考如下：

```
/* (non-PHPdoc)
 * @see CExtModule::autoloadProfile()
```

```
 */
protected function autoloadProfile()
{
	$this->merger2autoload('extend.cache.config', 'autoload.cfg.php');
}
```

[Code 4.1.2.2 扩展类入口程序之 autoloadProfile()]

[Code 4.1.2.2]中的 autoloadProfile()，使用了基类的$this->merger2autoload()函数，实现了载入需要自动加载的私有依赖类列表 autoload.cfg.php，此文件保存路径在/SPFW/extend/cache/config 目录下。

```
return array
(
'CFileCache'=> //文件缓存类  FileCache
	array('package'=>'extend.cache.lib',	'file'=>'CFileCache.class.php'),
'CMemcache'=> //分布式内存缓存类 CMemcache
	array('package'=>'extend.cache.lib',	'file'=>'CMemcache.class.php'),
'ICacheEngine'=> //文件缓存引擎接口
	array('package'=>'extend.cache.lib',	'file'=>'ICacheEngine.php'),
);
```

[Code 4.1.2.3 自动加载类配置 autoload.cfg.php]

[Code 4.1.2.3]就是 CCache 类所需的所有私有依赖类的文件配置路径以及类名，当系统需要用未加载的类时，会根据这个数组找到类文件的路径并自动完成加载，因此这个加载过程非常高效，大大降低了开发团队的项目维护难度（在使用类前无需考虑这个类是否加载的问题，类的加载是由模块或框架的设计者来维护的）。

setName()抽象函数，此为固定格式用于告诉基类当前类的名字，在父类自动生成日志的时候会用到。

```
/*
 * (non-PHPdoc)
 * @see CExtModule::setName()
 */
protected function setName($sModule=__CLASS__)
{
	$this->msModule = $sModule;
}
```

[Code 4.1.2.4 扩展类入口程序之 setName()]

isAbleRun()抽象函数：此函数的功能是检查当前运行环境是否符合要求（如 PHP 版本不够，或 PHP 的某个函数未安装，或某个目录是否有可写权限），用于模块自动化检查。下面是 CCache 模块中的样例代码：

```
	/* (non-PHPdoc)
	 * @see CExtModule::isAbleRun()
	 */
	public function isAbleRun()
	{
		$aRet = array();
		if (class_exists('Memcache'))
			$aRet['CMemcache (Memcache)'] = true;
		else
			$aRet['CMemcache (Memcache)'] = false;

		$sPath = getMAC_ROOT(). getFW_ROOT() . strtr(CFileCache::msCACHE_PACKAGE, array('.'=>'/'));
		if (!CFileOperation::creatDir($sPath)) //创建缓存目录
			$aRet['CFileCache (Write permissions: '. $sPath .')'] = false;
		if ((0x02 & CFileOperation::file_mode_info($sPath)) === 0)
			$aRet['CFileCache (Write permissions: '. $sPath .')'] = false;
		else
			$aRet['CFileCache (Write permissions: '. $sPath .')'] = true;
		return $aRet;
	}
```

[Code 4.1.2.5 扩展类入口程序之 isAbleRun ()]

[Code4.1.2.5]中检查了 php 的 memcache 模块是否安装，以及检查缓存目录是否有可写权限。以上的所有 CExtModule 中的抽象函数都在模块类中实现后，再继续编写模块中自己的逻辑。完成模块的开发后，外部调用方式如下：

```
$oC = new CCache();//邮件对象
```

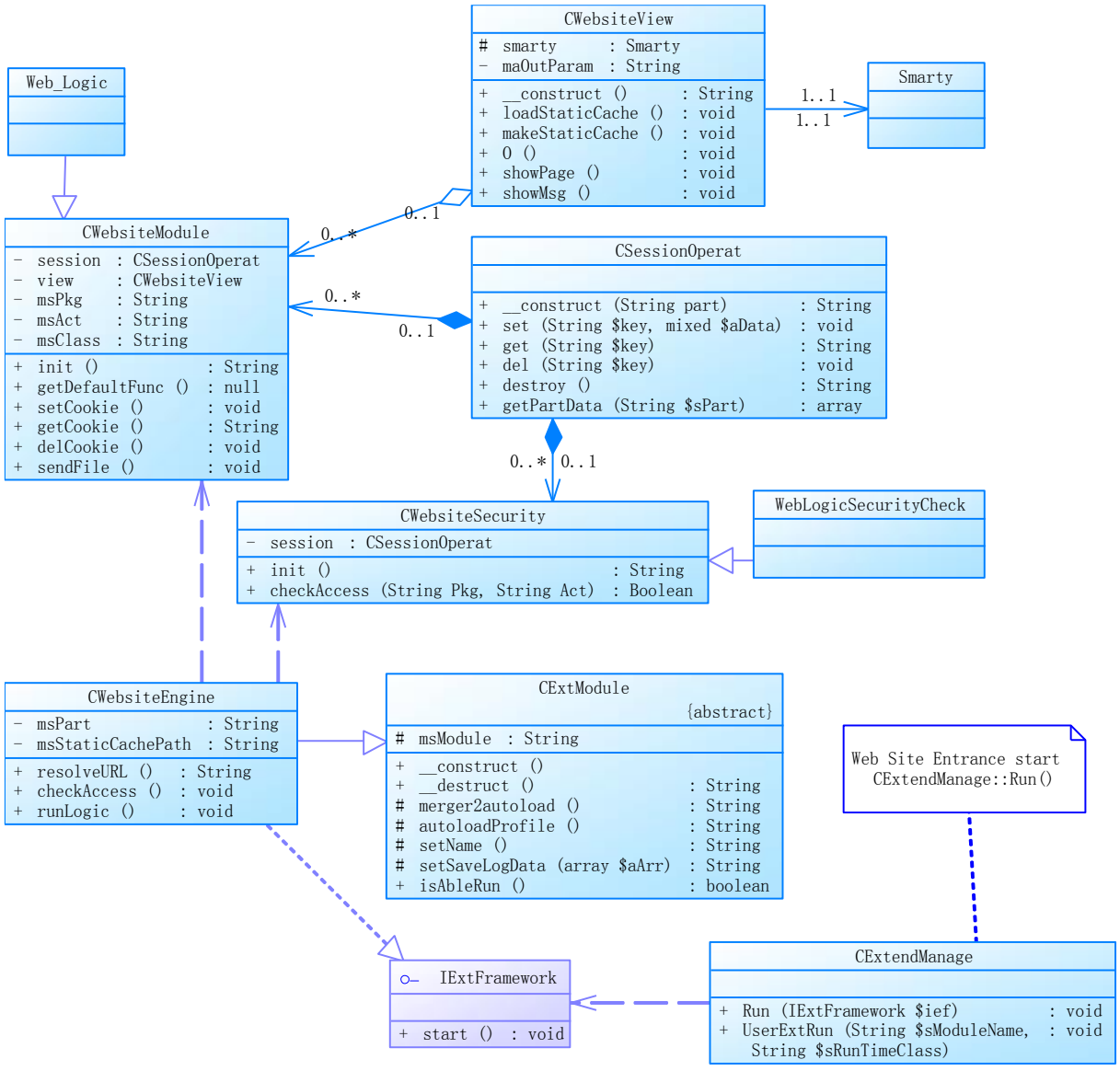
[Code 4.1.2.6 扩展模块 CCache 的调用]

4.1.3 扩展框架的开发

框架是用来管理某一种应用的设计模式，其价值是管理与组织所有业务层的类，让所参与业务逻辑开发的人员按照某一套标准的接口来实现业务逻辑，让项目的开发更易于管理。所以框架一般都有一个入口程序，比如 website_engine 就由一个入口点开始（即[Code 2.4.3.1]示例的代码），访问网站提供的所有业务逻辑功能。

因此框架的开发与扩展模块的开发一样，只是在这个基础上需要实现 IExtFramework 这个接口，实例化一个入口点，接口函数是 start()。此处代码较多可直接参考 SPFW/extend/website_engine/runtime/CWebsiteEngine.class.php 这个框架的源代码中的 start()函数。

我们通过 website_engine 类图来了解一下扩展框架的启动方式：



[Code 4.1.3.1 website_engine 框架的类图]

[Code 4.1.3.1]中左下角的 CwebsiteEngine 类为运行时的入口类，它继承了 CExtModule，同时实现了 IExtFramework 接口。最终在使用时入口文件会通过 CExtendManage 类的 Run()方法来启动，这也就是我们在[Code 2.4.3.1]中看到的 CExtendManage::Run(new CWebsiteEngine('user'));方法的由来。

4.1.4 项目的私有类编写

在项目的开发中，常需要开发一些在本项目内使用的公共类，但这些类不具备通用性。如果有这样的需求可以把项目的这些私有公共类放到/SPFW/extend/private_class/lib，并配置/SPFW/extend/private_class/config/autoload.cfg.php 文件设置类的存放路径。接着在需要使用的地方调用 CExtendManage::Run(new CPrivateClas())加载扩展类（一般在入口文件内创建，或在业务逻辑内的构造函数中加载，因为只需要加载一次）。加载完后就可以使用 new 的方式来创建定义的私有类。