【一】插件定义

插件是独立于系统内置功能以外的扩展功能，可围绕云引擎系统使用插件完成各种个性化需求。

云引擎后台带有插件管理，通过在系统中预设的各类钩子HOOK□实现插件与系统的对接，包括插件的安装、卸载、开启停用、设置等。

【二】插件目录

每一个应用插件都是一个独立的插件目录，所有的应用插件都是存放在appPlugin目录下。

以下为一个短信发送示例插件 MySmsDemo□ 插件code标识为 MySmsDemo □□

以下示例是一个插件的目录结构：

```
├── app
│ ├── Plugin [所有插件总目录]
│ │ ├── MySmsDemo[插件目录]
│ │ │ ├── Contracts [接口目录，可选]
│ │ │ │ ├── MessageInterface.class.php [短信内容接口类]
│ │ │ │ ├── PhoneNumberInterface.class.php[手机号接口类]
│ │ │ │ ├── SmsInterface.class.php[短信发送接口类 规范必须实现send 方法]
│ │ │ ├── Controller [控制器目录，需要提供外部URL访问的时候才需要，可选]
│ │ │ ├── Service [服务目录]
│ │ │ │ ├── Sms.class.php [短信发送实现类，对外开放的方法send□其中send会调用插件的send方法]
│ │ │ │ ├── MySmsDemo.class.php [具体的实现发送短信类 implements SmsInterface ]
│ │ │ ├── Support [公共目录]
│ │ │ │ ├── Message.class.php [Message implements MessageInterface接口]
│ │ │ │ ├── PhoneNumber.class.php [PhoneNumber implements PhoneNumberInterface接口]
│ │ │ ├── View [视图目录]
│ │ │ │ └── config.html[插件自定义后台设置页，可选]
│ │ │ └── MySmsDemoPlugin.class.php [插件定义和实现的文件，必须实现]
│ │ │ └── config.php [配置文件，有配置项的话，可选]
```

【三】插件命名

- 插件最外层文件夹和插件标识一致，使用大驼峰命名方式，插件标识唯一；
- 插件code标识必须在应用市场唯一，在阿帕云首次提交应用时，系统会自动提示是否重复；

【四】入口文件说明

如上例MySmsDemoPlugin.class.php文件为插件的入口文件，命名格式为：插件标识+Plugin.class.php

入口文件必须继承 Plugin.class.php 文件中的 Plugin 类；

类型 名称 名称

类属性 $code 插件标识，需和发布插件填写的标识一致

类属性 $hooks 插件调用钩子，数组格式，至少需要实现一个钩子

类属性 $custom_config 自定义插件设置页模板，默认值config.html

方法 install() 插件安装程序，返回值必须是bool类型。

方法 uninstall() 插件卸载程序，返回值必须是bool类型。

方法 hook_name(&$params) 如果定义了插件调用钩子，则需实现对应钩子方法，方法名称同钩子名称

代码示例

```
<?php
namespace Plugin\MySmsDemo;
use Plugin\MySmsDemo\Service\Sms;
use Plugin\Plugin;
class MySmsDemoPlugin extends Plugin
{
    /**
    * 插件标识
     * @var string
     */
    public $code = 'MySmsDemo';
    /**
    * 插件信息
     * @var string
     */
    public $info
        = array(
            'name'        => 'MySmsDemo',
            'title'       => '我的短信接口',
            'description' => '这是一个短信插件示例',
        );
    /**
    * 插件挂载钩子
     * @var array
     */
    public $hooks = ['sms_instance', 'plugin_config'];
    /**
```

```
     * @var string 自定义设置模板
     */
    public $custom_config = 'View/config.html';
    /**
    * 安装程序
     * @return bool|mixed
     */
    public function install()
    {
        $smsConfig = require ROOT_CONFIG_PATH . 'smsConfig.php';
        $smsConfig[$this->code] = [
            'className'   => $this->code,
            'channelName' => '我的短信(插件)',
        ];
        $this->updateConfig($smsConfig);
        return true;
    }
    /**
    * 卸载程序
     * @return bool|mixed
     */
    public function uninstall()
    {
        $smsConfig = require ROOT_CONFIG_PATH . 'smsConfig.php';
        if (isset($smsConfig[$this->code])) {
            unset($smsConfig[$this->code]);
        }
        $this->updateConfig($smsConfig);
        return true;
    }
    private function updateConfig($config)
    {
        $smsConfigFile    = ROOT_CONFIG_PATH . 'smsConfig.php';
        $smsConfigContent = "<?php\nreturn [\n";
        foreach ($config as $key => $item) {
            $key              = is_int($key) ? $key : "'$key'";
            $smsConfigContent .= $key . "=> [\r";
            $smsConfigContent .= "'className'   =>
'{$item['className']}',\r";
            $smsConfigContent .= "'channelName'   =>
'{$item['channelName']}',\r";
            $smsConfigContent .= "],\r";
        }
        $smsConfigContent .= "];";
       // 覆盖配置文件
        file_put_contents($smsConfigFile, $smsConfigContent);
    }
    /**
    * 插件业务程序
     * @param mixed $params
     */
```

```
    public function sms_instance(&$params)
    {
        if ($params == $this->code && $this->verify()) {
            $config = $this->getConfig();
            $params = new Sms($config);
        }
    }
    /**
    * 插件业务程序
     * @param mixed $params
     */
    public function plugin_config(&$params)
    {
        if ($params['code'] == $this->code && $this->verify()) {
            $template   = M('template')->where([
                'type' => [
                    'in',
                    '1,2',
                ],
            ])->field('id, code, title')->order('sortId asc, id
asc')->select();
            $templateDb = [];
            foreach ($template as $item) {
                $templateDb[$item['id']] = [
                    'code'  => $item['code'],
                    'title' => $item['title'],
                ];
            }
            if (!empty($params['config']['TemplateCode']['value'])) {
              // 这里因为索引是数字，不能用 array_merge
                $params['config']['TemplateCode']['value'] =
$params['config']['TemplateCode']['value'] + $templateDb;
            } else {
                $params['config']['TemplateCode']['value'] = $templateDb;
            }
            $params['title'] = '我的短信';
        }
    }
}
```

【五】后台设置页面

插件不一定有设置页面。必须插件目录里有一个config.php并且返回一个非空的数组才会显示。设置页里有显示。以示例插件的配置文件举列：

```
<?php
return [
    'apiId'      => [
        'title' => 'ApiId:',
        'type'  => 'text',
        'value' => '',
    ],
```

```
    'apiKey'  => [
        'title' => '密钥(ApiKey)',
        'type'  => 'textarea',
        'value' => '',
    ],
    'signature'  => [
        'title' => '签名(Signature)',
        'type'  => 'text',
        'value' => '',
    ],
    'TemplateCode'  => [
        'title' => '模板编号',
        'type'  => 'text',
        'value' => '',
    ],
];
```

如果插件配置比较复杂需要自定义页面, 则Demo插件的定义类需要挂载plugin_config钩子和定义$custom_config属性，详见上方 入口文件-》代码示例

控制器

插件控制器是根据前后台模块继承不同的controller, 如用户后台继承UserController等， 当URL中传入插件控制器变量的时候，会自动定位到插件控制器中的操作方法。例如我们在URL中传入：http://serverName/User/Payment/index/plugin/Demo

调用Demo插件（位于app/Plugin目录下面）的PaymentController控制器，文件位于app/Plugin/Demo/Controller/PaymentController.class.php,插件控制器本身的定义和普通的访问控制器一样，例如：

```
namespace Plugin\Demo\Controller;
use Common\Controller\UserController;
class PaymentController extends UserController
{
    public function index(){
        echo 'Plugin Demo';
    }

}
```