SDK3.0的实现机制

last update 2/22/11

目录概述

```
|- helpers/
|- modules/
|- vendors/
|- SDK (php file)
```

• helpers

这里放静态类，如Arr / Text等等，可以拿来直接用，不需要进行任何配置

```
Arr::overwrite($arr1, $arr2);
Text::flat('hello world');
```

• modules

所有的模块都放在这里，如http / cache / log等等。

一个典型的module文件夹结构：

```
|-cache
      |- cache.php
      |- exception.php
      |- adapter.php
      |- adapter/
            |- file.php
            |- memcache.php
      |- config/
            |- cache.php
```

• 类文件命名

- 如果是入口文件(如cache/cache.php)，直接命名为Class Cache

- 默认按照”文件夹_文件名”命名，如adapter.php命名为 Cache_Adapter

• 配置文件

- 如果有配置文件的话，路径均为 config/modulename.php
- 对配置项的注释都写在行尾
- 配置文件的内容书写规则
- 配置项的key为类名
- 配置项的key跟变量书写规范一致：驼峰法

```
return array(
      ‘Cache’ => array(
            ‘adapter’ => ‘Cache_Adapter_Memcache’,
      ),
      ‘Cache_Adapter_Memcache’ => array(
            ‘host’ => ‘localhost’, // host
            ‘port’ => 11211, // port
            ‘compress’ => TRUE, // compress?
            // ... other config
      ),
);
```

• 适配器模式

如果可以有多种媒介去完成同一件事情，如Cache可以有多种存储方式，Log可以有多种记录方式，抽象类统一命名为adapter.php，实现类统一放在adapter文件夹下。

• 异常处理

如果一个模块要抛出异常，必须继承SDK_Exception类，如

```
Class Cache_Exception extends SDK_Exception {}
```

• 入口类

入口类(文件名与模块名相同的那个php文件)必须继承SDK_Base

```
Class Cache extends SDK_Base {}
```

• vendors

这个文件夹里放的都是第三方类，如firephp / doctrine-dbal / doctrine-orm 等等，部分模块可能依赖这些第三方类库。

系统模块(SDK)

系统模块一共包含三个文件

- sdk.php
- base.php
- exceptin.php

• sdk.php

SDK类主要包含以下几个方法

• factory

全局工厂方法，如

```
$cache = SDK::factory('Cache');

// 第二个参数相当于factory的key，可以根据这个key返回不同的对象
$cache = SDK::factory('Cache', 'Cache_Adapter_Memcache');

// 可以将config作为第三个参数传递给构造函数
// 如果不传的话，将会读取指定的config文件
$cache = SDK::factory('Cache', 'Cache_Adapter_File', $config);
```

• instance

完全同上，只不过如果该类已经存在的话，会返回之前的类

• setConfigDir

设置config文件的文件夹路径，如果不设置的话，则读取模块默认的config文件

```
// 以后无论是SDK::instance或SDK::factory都会从该文件夹读取config
SDK::setConfigDir('/path/to/config');
```

- **initModules**

初始化模块，如果模块存在init.php文件，则在SDK::init时会载入该init.php文件

- **getConfig**

获取config，如

```
// key为类名，一般不需要显式获取config
SDK::getConfig('Cache');
SDK::getConfig('Cache_Adapter_File');
```

- **autoload**

设置SDK的自动载入

• base.php

所有的入口类都要继承SDK_Base类，该类主要有以下几个作用

- **getter / setter**

```
class Cache
{
	protected $_adapter;

	public function getAdapter()
	{
		return $this->_adapter;
	}
	public function setAdapter($adapter)
	{
		$this->_adapter = $adapter;
	}
}
$cache = SDK::instance('Cache');
$cache->adapter = 'Cache_Adapter_Memcache';
echo $cache->adapter;
```

- **config**

继承了SDK_Base后，就有了统一的config获取方法

```
class Cache extends SDK_Base
{
	//$config可能是用户手动传的或从config文件获取的
	// 在正式设置$config之前可以对传入的$config做些设置
	protected function _beforeConfig($config)
	{
		//...
	}

	// config设置完后，可以对$this->_config做进一步处理
	// 如对某些配置项进行验证之类的
	protected function _afterConfig()
	{
		//...
	}

	// config配置完后，要执行的一些操作
	protected function _afterConstruct($id)
	{
		//...
	}
}
```

• **extend**

```
class My_Cache
{
	public function foo(&$parent, $param)
	{
		// do something with $parent
		// here you can access $parent's protected method
	}
}

$cache = SDK::instance('Cache');
$cache->attach_behavior('My_Cache');
// $cache now has 'foo' method
$cache->foo('hello world');
```

其他要点

• 版本号

每个模块/助手类都有版本号，版本号的规则如下

版本的制定标准为: a.b

如果类的某些方法进行了内部重构,则升级版本号b

如果类新加了方法,则升级版本号a

如果类的接口发生改变,则将类重新命名为ClassX，也就是新建一个模块。

也就是说,一个模块一旦形成,就只能新增方法或调整内部实现。

• 注释

- 使用英文注释
- 类和方法的注释可以包含demo
- 类/方法的注释部分，如果不是"@"开头的标签，统一使用markdown语法
- 无论是公开的还是私有的方法都要加注释

- 新增的method，加上”@since” 标签
- @param type $param intro (如 @param string $className class name)
- 如果@param类型不定，则为”mixed”

• PHP内部数据类型

如 true / false / null 统一都小写

• 异常

抛出异常时，如果有变量，则用”{}”括起来，作为第二个参数的key

```
throw new Cache_Exception(‘Cache dir {dir} is not writeable’,
      array(‘{dir}’ => $path)
);
```