PHP Dever框架文档

rabin

目　录

序言

Dever是一个PHP框架，为方便服务化、接口化、大前端化而实现的后端框架。

初次使用

核心理念

在Dever里，我们通常把项目称之为DeverApp（或称组件、项目）。一个大项目很可能由多个DeverApp组成。

业务类库：每个组件都有自己的核心类库，用于实现该组件的具体功能。请尽量遵循设计模式来开发。

数据模型：与数据库交互的类或者配置文件，通过调取数据模型，可以快速的从数据库中获取数据。

数据绑定模板引擎：高效的且创新的模板引擎，同时支持传统的（类smarty）模板引擎与具有Dever特色的数据绑定模板引擎，更为方便前端使用。

基础知识

项目目录

一个DeverAPP由以下部分组成：

名称	描述
index.php	入口文件
assets	模板原始文件，如果你这个DeverApp是一个页面展示项目，必须有此目录。可以在配置中更改这个路径，改到另外的一个统一的目录里去，具体说明参考模板引擎
template	模板数据绑定文件，页面展示项目，必须有此目录。和assets里的模板一一对应，具体说明参考模板引擎
config	配置文件，大部分配置如host、db等都在Dever的核心框架里配置完成，这里的配置是对本DeverApp做的独有配置，具体使用方法参考配置索引
src	类库源码，你的所有业务逻辑都写在这里，具体开发方法参考业务类库，该目录名可以更改。
	数据文件目录，里面包含有模板编译文

data	件、上传文件、缓存文件、项目配置文件、数据库表结构文件等，开发环境下可拥有777权限，生产环境下处上传文件外其他目录建议只读，该目录位置可以根据配置更改。
database	数据库模型文件目录，通过配置生成数据表及数据模型，也可以借此生成通用后台，通用后台需要安装manage组件

入口文件

入口文件的名称不一定是index.php，一个基本的入口文件中应做如下定义：

```
//$_SERVER['DEVER_SERVER'] = 'setting';
//define('DEVER_PROJECT', 'default');
//define('DEVER_PROJECT_PATH', dirname(__FILE__) . DIRECTORY_SEPARATOR);
//define('DEVER_ENTRY', 'index.php');
define('DEVER_APP_NAME', 'cms');
define('DEVER_APP_LANG', '文章管理系统');
define('DEVER_APP_PATH', dirname(__FILE__) . DIRECTORY_SEPARATOR);
include('common/dever_v1/boot.php');
```

下面我们一步一步对上述代码做一下说明。

1. 初始化DeverApp的环境：

```
$_SERVER['DEVER_SERVER'] = 'setting';
```

DEVER_SERVER是Dever框架的环境变量，可以在nginx里设置，也可以在入口文件中设置，如果不设置这个环境变量，Dever将自动读取当前域名的跟域名作为它的值。它的用处是，用以区分不同环境下的项目配置，在config的env目录下，可以建立不同环境变量的目录。可以不定义该项。默认为default。请在condenast的项目中定义为setting，同时修改config/setting中对应的配置。

2、定义DeverApp所属项目：

```
define('DEVER_PROJECT', 'default');
define('DEVER_PROJECT_PATH', dirname(__FILE__) . DIRECTORY_SEPARATOR);
```

这两个常量用于定义当前的DeverApp属于哪个项目，如果和另外一个DeverApp属于同一个项目，我们的data目录的位置将有所变化，这个后续会做说明。暂时我们一个DeverApp就是一个项目吧。一般一个项目是不需要

定义这两项的。该定义将影响config/setting中的配置文件名。

3、定义DeverApp基本信息：

```
define('DEVER_ENTRY', 'index.php');
define('DEVER_APP_NAME', 'cms');
define('DEVER_APP_LANG', '文章管理系统');
define('DEVER_APP_PATH', dirname(__FILE__) . DIRECTORY_SEPARATOR);
```

这里做个表格说明：

名称	描述
DEVER_APP_NAME	DeverApp名
DEVER_APP_LANG	DeverApp中文名或者描述
DEVER_APP_PATH	DeverApp路径
DEVER_ENTRY	入口文件，不定义则为index.php

业务类库

基本理念

在这里实现类，如果想引入第三方类库，可以直接使用composer引入。

命名空间

Dever遵循psr-4协议，命名空间使用目录名定义。

类名

类名使用文件名，首字母大写来定义。

类方法

例子

假如我们建立了一个名为cms的DeverApp。

我们在cms下实现一个类为Cms\Src\News，这个类下有个方法为get，有两个参数，该文件保存在cms/src下。

代码如下所示：

```
<?php
namespace Cms\Src;

use Dever;

class News
{
    public function get($id, $name)
    {
        return $id;
```

```
    }

    public function get_api($id, $name, $callback)
    {
        return $callback;
    }

    public function get_secure_api(){}
}
```

调用

get方法有如下三种形式进行调用：

```
1、Dever::load('cms/service/news.get', 1, 2);
2、Dever::load('cms/service/news')->get(1,2);
3、$news = new \Cms\Service\News;$news->get(1,2);
```

如果我们还有个DeverApp，名为cat，想调用cms下的上述方法，只需用这三种调用形式即可。如果cat和cms没有部署到一台机器上，两者之间需要通过http协议（通过配置，也可以是tcp协议）来访问。这里无需修改，只需修改配置即可。详情见配置索引中的项目配置。

外部访问

如果该方法想被浏览器（http协议）直接访问，可以在方法名上加_api后缀。如get_api。如果之前实现过get方法，不想改变原有代码，可以直接在类下增加get_api方法。

```
public function get_api($id, $name, $callback)
{
    return $callback;
}
```

其中，get_api中的参数$id, $name继承get方法中的参数，$callback是get方法的返回值。
在浏览器中，这样访问：cms/index.php?service/news.get?id=1&name=test
注意：方法名为main，则默认可以为外部访问。无需带上_api后缀。

访问加密

get_secure_api方法是加密处理get_api方法。一般不用实现函数体，dever会自动处理加密解密功能，加密算法可以参考后边的公共类。

数据库事务

在方法名中增加_commit后缀，会自动将该方法中的所有操作都加入到事务中：

```
public function get_commit()
{
    ...
}
```

数据模型

基本理念

从数据库中读取数据可以通过数据模型来处理。

Dever中的数据模型无需实现类，只需在database中实现配置文件即可。如果你只想通过sql的形式来读取数据库，那么database配置文件也是无需建立的。

当然，如果你想用类来实现数据模型，也是可以的，请参考对象数据模型。

开发准备

开发目录

我们在DeverApp中的database目录中开发。该目录中的文件名即为数据表名称。文件名小写。

建立新表

如果我们希望建立一个新表，比如新闻表：news，那么我们在开发目录下，建立一个news.php的文件即可。

基本属性

名称	描述
name	表名
desc	描述
lang	表中文或英文描述，显示在通用后台
struct	数据结构配置，见下述数据结构表格说明
alter	更新表结构，见下述更新表结构说明
manage	针对通用后台的一些配置，见下述通用后台配置表格说明
request	表的方法定义，见下述表方法表格说明
sync	同步更新另外一个或者多个表的数据
start	插件钩子，当入库之前，自定义的操作
end	插件钩子，当入库之后，自定义的操作
check	检测字段唯一性

数据结构

如果你使用sql语句来读取数据库数据，无需定义本属性。

数据结构的定义属性名为struct，该项的子属性为多维数组，key就是字段名，例子如下：

```
# 数据结构
'struct' => array
(
    'id'        => array
    (
        'type'      => 'int-11',
        'name'      => 'ID',
        'default'   => '',
        'desc'      => '',
        'match'     => 'is_numeric',
        'order'     => 'desc',
    ),

    'name'      => array
    (
        'type'      => 'varchar-30',
        'name'      => '接口名称',
        'default'   => '',
        'desc'      => '请输入接口名称',
        'match'     => 'is_string',
        'update'    => 'text',
```

```
        'search'    => 'order,fulltext',
        'list'      => true,
    ),

    'state'     => array
    (
        'type'      => 'tinyint-1',
        'name'      => '状态',
        'default'   => '1',
        'desc'      => '请选择状态',
        'match'     => array('is_numeric', 1),
        'option'    => $option,
        'update'    => 'radio',
        'list'      => true,
    ),

    'cdate'     => array
    (
        'type'      => 'int-11',
        'name'      => '录入时间',
        'match'     => array('is_numeric', DEVER_TIME),
        'desc'      => '',
        # 只有insert时才生效
        'insert'    => true,
        //'list'        => 'date("Y-m-d H:i:s", {cdate})',
    ),
)
```

具体的说明见下表：

名称	描述
type	字段类型
name	字段中文名
default	字段默认值
desc	字段描述
match	数据入库或者查询时使用该项进行检测，可以为php函数，也可以为正则表达式
order	使用该字段进行默认排序
update	针对通用后台的配置，有此项代表该字段出现在后台更新页面，值为input的type类型
option	针对通用后台的配置，当update值为select、radio、checkbox时，该项有

	效，值为数组，可选项的所有可选参数
list	针对通用后台的配置，有此项代表该字段出现在后台列表页面，值为true则直接使用该字段的值，也可以直接使用{字段名}来更换显示的值，可以是php函数
list_name	针对通用后台的配置，当list有效时，如果不配置list_name，则列表显示的表头名字为name的值，否则显示list_name的值
insert	针对通用后台的配置，值为true时，只有插入才生效
search	针对通用后台的配置，有此项代表该字段出现在后台列表页面的搜索项中，目前有三个值：select（下拉选项）、order（排序）、fulltext（模糊匹配）

如果需要展示其他表的字段，请在struct里加上如下参数：

```
# 测试关联其他表字段 项目-表-字段
'manage-admin-username'     => array
(
    # 必填，两个表之间的关联字段，第一个为当前表的，第二个为关联表的
    'sync'      => array('id', 'id'),
    'default'   => '',
    'name'      => '测试其他表的字段',
    'desc'      => '测试其他表的字段',
    'match'     => 'option',
    'update'    => 'text',
    'list'      => true,
),
```

调取方式

使用db方法调取

获取数据模型

```
# 通用的方法
$news = Dever::db('cms/news');
# 如果想获取另外一个数据库配置，old是在配置索引中配置的数据库配置
$news = Dever::db('cms/news:old');
```

使用默认方法

```
$news->one(1);//传入id，查询一条数据
$news->all(array('option_id' => 1));//传入id，查询多条数据
$news->list(array('option_state'=> 1));//传入状态，查询多条数据，并且进行分页，分页请查看分页类
$news->insert(array('name' => 'test'));//插入数据
$news->update(array('name' => 'test', 'where_id' => 1));//更新数据
```

使用自定义方法

请在news.php中建立request属性，例子如下：

```
'request' => array
(
    'getAll' => array
    (
        'where' => array
        (
            //'username' => '/^([A-Za-z0-9])/',
            'config' => 'yes',
            'state' => 1,
        ),
        'type' => 'all',
        'order' => array('id' => 'desc'),
        'limit' => '0,5',
        'col' => '*',
    ),

    'getOne' => array
    (
        'type' => 'one',
        'where' => array
        (
            'id' => 'yes',
            'config' => 'yes',
            'state' => 1,
        ),
        'input' => true,
    ),
),
```

其中"getAll"和"getOne"均为自定义的方法名，可以直接使用$news->getAll($param)。

具体的说明见下表：

名称	描述
type	方法类型，目前有one、all、update、insert、delete五种方法

order	排序
limit	获取多少条数据
col	获取的字段
input	为true则可以从http中的$_GET、$_POST获取值
where	where条件的必填项，值为字段名，字段的值为入库前要进行的验证，如果是yes，则直接继承match里的值，如果要用in等特殊方法，可以这样：'id' => array('yes', 'in')，如果需要定义字段的值，在yes后面加上即可：yes-colname
option	where条件的选填项
set	update语句的更新项
add	insert语句的更新项

联表查询

在上述的自定义方法中，增加join项，即可进行联表查询，如：

```
'getAll' => array
   (
       'where' => array
       (
           'config' => 'yes',
           'state' => 1,
           't_2.cate_id_in' => array('yes-t_2.cate_id', 'in'),
           't_2.cate_id' => 'yes',
       ),
       'join' => array
           (
               array
               (
                   'table' => 'service/company',
                   'type' => 'left join',
                   'on' => array('company_id','id'),
                   'col' => 't_2.id as cid',
               ),
           ),
       'type' => 'all',
       'limit' => '0,5',
       'order' => array('t_2.sdate' => 'desc','id' => 'desc'),
       'col' => 'id,t_1.name,t_1.content,t_2.name as company_name',
   ),
```

注意，联表查询里的表名已经自动变为缩写，如主表为t_1，连接的第一个表为t_2，以此类推。

使用sql

```
$news->query('select * from {table} where id = :0', array(':0' => 1))->fetch();
//获取一条数据
$news->query('select * from {table} where id = :0', array(':0' => 1))->fetchAll(
);//获取多条数据
$news->fetchAll('select * from {table} where id = 1');//获取多条数据
$news->fetch('select * from {table} where id = 1');//获取一条数据
$news->fetchAll('select * from {table} where id = :0', array(':0' => 1), array(
'num' => 10));//获取多条数据并进行分页，分页请查看分页类
```

使用load方法调取

以上所有方法，都可以通过：Dever::load('cms/news')来替换。或者直接使用Dever::load('cms/news-one')来调取。

该方法会自动通过load缓存机制进行缓存，具体的缓存说明请参考配置索引中的缓存项。

同步更新

```
# 同步更新另外一个或多个表的数据
    'sync' => array
    (
        'manage/admin_role' => array
        (
            # 更新另外一个表的字段 => 本表的字段
            'where' => array('admin_id', 'id'),
            # 要更新的数据
            'update' => array('role_id' => 'role'),
            # 同步更新的类型，delete为先删再插入，update为直接更新
            'type' => 'delete',
        )
    ),
```

检测唯一性

```
# 检测email必须唯一
'check' => 'email',
```

当值为数组时，如array('email', 'name')，则分别检测这两个字段的唯一性。（或）

当值为字符串并且有逗号隔开时，如'email,name'，则将两个字段的值合并一起检测唯一性。（与）

对象数据模型

如果想使用类来实现数据模型、操作sql的话，请在数据库配置中增加一项：

```
'compatible' => 'model',
```

model为database目录下的子目录，建立该目录时，首字母大写。

```
database/Model/News.php
<?php
namespace Cms\Database\Model;
use Dever;

class News
{
    private static $name = 'news_list';

    public static function all($param)
    {
        $data = Dever::db('old')->fetchAll('select * from ' . self::$name . ' l
imit 1');

        return $data;
    }
}
```

调用方法和上述一致即可：

```
Dever::db('cms/news')->all($param);
```

事务

在业务类库中，使用数据模型的时候，可以使用事务，功能与业务类库中的_commit后缀相同。例子如下：

```
    public function test_api()
    {
        $cat = Dever::db('demo/cat');
        try {

            $cat->begin();

            $data['name'] = '服饰' . time();
            $id = $cat->insert($data);

            echo $id ."\r\n";

            $data['name'] = '潮流' . time();
```

```
            $data['id'] = 1;
            $id = $cat->insert($data);

            echo $id ."\r\n";
            $cat->commit();
        } catch (\Exception $e) {
            $cat->rollBack();
        }

        $data = $cat->getList();
        echo Dever::sql();
        print_r($data);die;
    }
```

模板引擎

基本理念

Dever里的模板引擎分为两种模型：数据绑定模型和传统模型。

开发者可以根据实际需要来选择一个模型来使用。

数据绑定模型

该模型仿照jquery，Dever把数据接口与DOM进行绑定，生成编译文件，输出页面。

优点：不影响前端模板基本结构。快速套接页面。如果做好前后端约定，非常方便改版。

缺点：上手慢，需要熟悉具体规则。个别dom层次较深的前端页面无法使用。

开发目录

在DeverApp的template下开发。需要定义assets（前端模版）目录。

例子

```
<?php

$view

->set('$title = Dever::dyna("home")->title')
->set('$Keywords = Dever::dyna("home")->keyword')
->set('$Description = Dever::dyna("home")->desc')
->set('Dever::$global["ckey"] = "stories"')

->loop('#sc1 ul li', 'main/home.focus_data', array
(
        'a|$v.child' => array
        (
            'href' => '<{$v1.news_link}>',
            'img|0' => array
            (
                'src' => '<{$v1.pic}>'
            )
        ),
        'p .channel_name' => array
        (
            'html' => '<{$v.cname}>',
            'href' => '<{$v.news_catlink}>'
        ),
))

->fetch('.fn-hide', '<{Dever::load("main/adbanner.homead")}>')
->display();
```

解析

1、设置页面内全局变量

```
->set('$title = Dever::dyna("home")->title')
```

$title 为变量名，Dever::dyna("home")->title为值，Dever为框架的公共类，具体方法请参考公共类说明。这之后可以直接使用<{$title}>来使用。

2、简单数据绑定

```
->fetch('.fn-hide', 'main/adbanner.homead')
```

.fn-hide为页面中的class名，这里完全仿照jquery，如果页面中有id，则可以使用#id名。默认绑定到该节点的内容中，如果要修改节点属性，请加上@符号，如.fn-hide@href

<{Dever::load("main/adbanner.homead")}>为接口地址，该接口可以直接通过浏览器访问，得到数据。如果是接口中对的某个字段，则可以写成<{Dever::load("main/adbanner.homead#name")}>，这里，也可以简写成'main/adbanner.homead'的形式。

当main/adbanner.homead返回的数据是多维数组时，可以写成main/adbanner.homead[0]的形式。

3、批量数据绑定

```
->loop('.ad', 'main/adbanner.homead', array
(
    'style' => array('height:70px;border-top:1px solid #eeeeee;', 'display:none
;'),
    'test' => '<{$v.name}>',
    'span|$v.data' => '<{$v1.name}>',
))
```

批量数据绑定，用于针对同一个dom同一个接口，做多个属性及其子dom的数据绑定操作。

需要注意的是，如果值为array(0,1)，则进行逻辑判断当前接口返回的值，如果存在，则取0，不存在则取1。

4、接口数据分配

```
->render('main/adbanner.homead', array
(
    '.topbar@style' => array('height:70px;border-top:1px solid #eeeeee;', 'none'
),
    '#menu@test' => '<{$v.name}>',
    '.topbar span|$v.check' => array('test', 'none'),
    /*
    '.topbar span' => array
    (
        'self|$v.data' => '<{$v1.name}>',
    )
    */
))
```

与“批量数据绑定”相反，这里可以把一个接口（返回数组或json）下的所有数据，绑定到当前页面下不同的dom上。

5、循环数据绑定

```
->loop('#sc1 ul li', 'main/home.focus_data', array
(
```

```
        'a|$v.child' => array
        (
            'href' => '<{$v1.news_link}>',
            'img|0' => array
            (
                'src' => '<{$v1.pic}>'
            )
        ),
        'p .channel_name' => array
        (
            'html' => '<{$v.cname}>',
            'href' => '<{$v.news_catlink}>'
        ),
))
```

如果需要循环一个节点，比如li节点，那么就要传入第三个参数，并且第二个参数只能为接口名。这里主要对第三个参数做说明：

- 第三个参数主要处理当前li节点下的子节点数据绑定。里面的变量为$k和$v
- 如果子节点需要再次循环，则在节点名后加上要循环的数据：a|$v.child，之后它里面的值为$k1、$v1，支持无限极循环。($kn、$vn)
- 当子节点有很多相同的名称时，可以使用|0、|1、|n来区分，如果有要循环的数据，需要写成a|$v.child|0。如果只需要main/home.focus_data中的其中几条数据，比如第0条到第1条，只需加上main/home.focus_data[0-1]即可。

6、逻辑判断

```
->loop('#test2 li', 'demo/vogue/news.data', array
(
    'span' => array
    (
        'if($v.id == 1)' => '$v.id = 3',
        'else' => '$v.id = 4',
        'endif' => 2,
        'html' => '$v.id'
    ),

    'if($v.type == 1)' => array
    (
      'li|$v.data1' => array
      (
        'span' => array
        (
            'html' => '$v1.id'
        ),
        'a' => array
```

```
            (
                'html' => '$v1.name'
            ),
        ),
    ),
    'else' => array
    (
        'li|$v.data2' => array
        (
          'span' => array
          (
              'html' => '$v1.id'
          ),
          'a' => array
          (
              'html' => '$v1.name'
          ),
        ),
    ),
    'endif' => 1,
    'a|0' => array
    (
        'html' => '$v.name'
    ),
))
```

endif代表if判断结束，如果值为1，则代表if中的所有变量不能为下面所用，如果为2，则可以为下面使用。

7、定义函数

```
->fetch('myfunc:#fetch:这是描述', 'demo/vogue/news.getData#id')
```

在任意一个绑定方法第一个参数中增加冒号，第一个为函数名，第二个为原来的dom，第三个参数为描述（可为空，主要用于给别人阅读），则生成一个公共的模板方法，只能在模板中使用：

```
->call('demo/manage/test:myfunc')
```

前端开发

前端可以在DeverApp的assets下开发。也可以将assets放置到任意位置，然后通过配置来定位assets目录。

在开发一个页面时，请在头部引入dever.js，例子

```
<!DOCTYPE html>
<html lang="zh-CN">
<head>
```

```
  <filter>
    <script src="http://ip/dever/script/jquery.min.js"></script>
    <script src="http://ip/dever/script/dever.js"></script>
    <script src="http://ip/dever/script/template.dever.js"></script>
  </filter>
  <include system="" path="inc/" file="head"></include>
</head>

<body>

<include system="" path="inc/" file="top"></include>

<div id="name">test</div>

<div id="list">
<ul>
<li><span>id</span>:<p>name</p></li>
</ul>
</div>

</body>
</html>

<dever>
//$("#name").fetch("demo/vogue/news.data[0]#name");
$("#list li").loop("demo/vogue/news.data", {
    "p" : {
        "html" : "$v.name"
    }
    ,"span" : {
        "html" : "$v.id"
    }
});
</dever>
```

加载公共文件

```
<include system="" path="inc/" file="head"></include>
```

system：DeverApp的名称，默认为空

path：路径

file：文件名

前端数据绑定

有些时候，不需要后端来负责做模板页面套接，后端只需出接口，所有的工作都前端完成，当前端由于各种原因无法使用异步加载（vue、nodejs等）来渲染页面时，可以使用dever.js提供的方式：

```
<dever>
//$("#name").fetch("demo/vogue/news.data[0]#name");
$("#list li").loop("demo/vogue/news.data", {
    "p" : {
        "html" : "$v.name"
    }
    ,"span" : {
        "html" : "$v.id"
    }
});
</dever>
```

基本语法和上述解析一节讲述的基本一致。请注意json参数的拼写正确。

传统模型

该模型仿照smarty，Dever直接把伪php代码进行编译，生成编译文件，输出页面。

优点：上手快，只要熟悉mvc模式、熟悉smarty，可以快速开发。

缺点：需要直接更改前端模版，破坏前端模版代码。前端改版后，需要重新套接页面。

开发目录

模版在DeverApp的assets目录开发。可以自定义目录。

控制器请在DeverApp的业务类库中开发。

例子

控制器

```
<?php
namespace Manage\Src;

class Api
{
    public function test_api()
    {
        $data['test'] = 1111111111111;
        $data['data'] = $this->loaddata();

        return Dever::render('example/render', $data);
    }

    public function loaddata()
    {
        $config = array
        (
```

```
            array
            (
                'id' => 1,
                'name' => '1111',
                'child' => array
                (
                    array
                    (
                        'id' => 3,
                        'name' => '3333',
                        'child' => array
                        (
                            array
                            (
                                'id' => 9,
                                'name' => '9999',
                            ),
                            array
                            (
                                'id' => 10,
                                'name' => '10101010',
                            )
                        ),
                    ),
                    array
                    (
                        'id' => 4,
                        'name' => '4444',
                    )
                ),
            ),
            array
            (
                'id' => 2,
                'name' => '2222',
                'child' => array
                (
                    array
                    (
                        'id' => 5,
                        'name' => '5555',
                    ),
                    array
                    (
                        'id' => 6,
                        'name' => '6666',
                    )
                ),
            )
        );
```

```
        return $config;
    }

    public function test_data()
    {
        return 1;
    }
}
```

模版

```
<div>
<{if($test):}>
<{$test}>
<{endif}>

<ul>

<{loop($data):}>
<li>
<{loop($v['child']):}>
<{$v1['name']}>
<{endloop}>
</li>
<{endloop}>

<{foreach($data as $k => $v):}>
<li>
<{foreach($v['child'] as $k1 => $v1):}>
<{$v1['name']}>
<{endforeach}>
</li>
<{endforeach}>

</ul>

</div>
```

配置索引

基本理念

Dever里的配置分为核心配置与项目配置两种。

核心配置：default.php

一般放置在核心框架的config目录下。核心配置请尽量让运维人员进行维护。

核心配置包括：数据库配置、host配置、缓存配置、debug配置等。

目录

在config目录下：

```
base.php
env/localhost/default.php
```

上述两个文件一般为某个项目的核心配置。

其中base.php为项目配置，可以暂时忽略。该配置会覆盖default.php中的配置。

env/localhost/default.php为核心配置。其中default.php为PROJECT_NAME的值，默认为default

配置中包含有几个类别。

默认已经将一部分default.php中不随着环境改变的配置放置在base.php中。

基本配置：base

名称	描述

lang	语言包设置
url	url默认参数，所有Dever::url生成的链接都会加上这个参数
urlEncode	开启url中某个字段加密
urlEncodeFilter	url的原始路径里包含什么字符，则不加密
urlEncodeMethod	使用加密解密的方法，如：array('Dever::idtostr', 'Dever::strtoid')
filter	是否启用自动过滤功能，必须加载manage项目，目前可选的值为：manage（自带的过滤功能，手动添加过滤词。非常简单，小型项目可以开启），bao10jie（必须申请账号），启用之后，需要在database的配置表文件中增加filter项
path	项目部署的相对路径
assets	访问assets目录的物理路径
data	访问data目录的物理路径
workspace	访问当前项目目录的物理路径
token	定义api的token明文
clearHeaderCache	是否启用nocache，如果是互动类的项目且主域增加了cdn，建议开启，也可以直接使用Dever::clearHeaderCache()方法
cron	开启用户触发cron，主要用于无法加到系统计划任务的虚拟主机，必须安装manage组件，谨慎开启，会稍微影响程序执行效率
api	api是否开启，默认关闭，如果开启，则需要在项目下建立api目录，手动指定api，类的方法后缀无需加上_api和_secure_api
apiDoc	api文档生成是否开启，开启后，将会根据访问来生成文档。生产环境建议禁止
apiLog	api日志是否开启，开启后，将会记录所有带有_api后缀方法的请求参数和响应参数

模板配置：template

名称	描述
assets	使用的模板 注意：定义这个之后，将会强制将本项目模板变成这个 定义成数组的话则为pc和手机版
path	模板所在目录（assets下），默认为html
template	定义这个之后，将强制将template下的子目录改为这个，不定义或不填写，则默认使用上边的template的值
strip	板编译时是否过滤\r\n
layout	是否启用layout 如启用，填写要替换的class或者id即可，具体效果可参考youtube，只加载部分内容，前端请加载pjax.js
replace	替换设置 一般用于替换资源，将模板中的（html中的）js等相对url换成绝对url，如果不定义，则默认为../js这样的，这是一个数组。
domain	是否启用静态资源域名动态化，启用之后，静态资源的域名将动态加载，适合使用多个域名或publish启用
shell	是否开启强制刷新页面缓存
name	是否开启手动更改模板名称，允许通过$_GET的方式来更改当前模板，值为$_GET的key值，默认关闭
publish	是否发布，此项开启后，系统不会检测service（意味着不用将service打包上线），适合生产环境，并能对代码起到一定的加密保护。

数据库配置：database

名称	描述
type	数据库类型，pdo、mysql、mongodb
host	主机地址，读写分离请添加read和update选项
database	数据库名

username	用户名
password	密码
compatible	数据模型的request方法兼容类
opt	是否开启sql自动优化，将sql中的select * 转换为 select a,b形式，将sql中的where条件按照索引从左到右自动排序，必须打开上述的opt选项，数据量大时建议打开。
sql	是否开启手动更改模板，当项目有多个模板时适用
create	关闭自助建表，生产环境建议开启，开启之后无法对数据表结构进行更新操作
other	定义另外的数据库，里面的项与上述配置项相同

缓存配置：cache

名称	描述
mysql	启用mysql数据库缓存，这个缓存是根据表名自动生成。无需手动添加
html	启用页面缓存 会根据当前的url来生成缓存，相当于页面静态化。
data	启用数据级别缓存 这个缓存是程序员自定义的：Dever::cache('name', 'value', 3600);
load	启用load加载器缓存
loadKey	缓存精细控制，可以根据缓存的key（mysql为表名、service为小写类名，规则是模糊匹配），来控制每一条缓存
shell	缓存清理的指令
expire	是否启用key失效时间记录，启用之后，将会记录每个key的失效时间
type	缓存类型
store	缓存保存方式，支持多个数据源、多台缓存服务器

调试配置：debug

名称	描述

error	开启错误提示 生产环境建议禁止
log	错误日志记录，为空则不开启
overtime	是否开启记录超时时间，单位为秒
request	开始访问报告,生产环境建议禁止或添加ip限制，多个ip用逗号隔开, 如禁止，值为false，下述shell也将失效,值为2，则开启强制模式，任何输出都将打印debug
shell	设定打印访问报告的指令

域名配置：host

名称	描述
base	根域名
cookie	cookie域名
assets	assets网络路径，会自动将assets替换为assets/模板
public	assets下的public目录
css	assets下的css目录
js	assets下的js目录
img	assets下的img目录
images	assets下的images目录
lib	assets下的lib目录
merge	合并之后的网络路径，填写之后自动合并资源，不填写则不合并，适合把资源放到云端
upload	上传系统的上传路径的域名(不带action)
image	上传系统的访问域名
proxy	是否启用代理功能，适用于跨域操作
api	内部接口定义，Dever::load将自动转为这个配置，适用于微服务
project	项目定义，Dever::load将自动转为这个配置，替换掉data/project/default.php里的数据

路由配置：route.php

项目配置一般放置在项目的config目录的route.php里。里面实现一个数组，key值为url地址，value为要定向的

业务类库。支持正则表达式。

如：

```
return array
(
    'home' => 'index',
    'default.html' => 'index',
    '([0-9]+)' => 'tag?id=$1',
    '([0-9]+)/index.html' => 'tag?id=$1',
    '([a-zA-Z_]+)/(.*?)/news_(.*?).html' => 'news?ckey=$1&time=$2&id=$3',
    '([a-zA-Z_]+)/(.*?)/news_(.*?)' => 'news?ckey=$1&time=$2&id=$3',
);
```

当使用Dever::url('tag?id=1');时，将自动生成链接：url/1/index.html

特殊动态配置：dyna.php

dyna可以定义seo，或者一些有趣的配置。

如：

```
return array
(
    'home' => array
    (
        'website'   => '1111111,$data[name]',
        'title'     => '11111',
        'keyword'   => '2222',
        'desc'      => '32313',
    )
);
```

调用方法：

```
$data['name'] = '测试';
$data = Dever::dyna('home', $data);
print_r($data['website']);
//或者这样 Dever::odyna('home', $data)->website;
```

公共类

- 使用数据绑定形式
- 使用传统的smarty形式

基本理念

Dever里的公共类名为Dever。为了方便使用，大部分功能都通过Dever类来调取。

在实现类的时候，直接使用use Dever即可调取Dever类库中的大部分方法。

配置与全局变量

读取配置

```
Dever::config('base')->name;//读取
Dever::config('base')->name = 1;//写入
```

设置全局变量

```
Dever::$global['name'] = 1;
```

调试相关

shell

shell是一种特殊的传参，一般用在get/post中，使用该方法可以自定义一些shell

```
if(Dever::shell('test'))
{
    echo 'yes';
}
```

上述的意思是，当shell的值中包含有test的时候为真。默认的shell有：

1、debug：开启调试功能，sql执行后的数据不输出，只输出数量。

2、clearcache：清理缓存

这两个shell的值都可以在配置中更改。

3、all：将调试功能中的sql执行后的数据也输出。

debug调试

```
Dever::debug('test');
```

sql调试

将输出上一条sql。如果传入参数为all，则输出所有sql

```
print_r(Dever::sql());
```

error日志记录

将输入的信息记录到log日志中

```
Dever::error('test');
```

输出相关

一般正常的输出，我们使用return $data;即可，如果需要异常处理等输出，请参考以下方法：

标准输出

输出的格式同return $data一致

```
Dever::out('你好');
```

自定义标准输出

输入什么，就输出什么

```
Dever::outDiy('你好');
```

警告提示

输出警告提示后，后面所有的代码都不能执行。

```
Dever::alert('你好');
```

分页信息

```
Dever::pageInfo();
```

输入相关

获取ip

```
Dever::ip();
```

获取输入参数

```
Dever::input('test');
```

设置输入参数

```
Dever::setInput('test', 'value');
```

获取带有某个前缀的所有输入参数

```
Dever::preInput('test');
```

http相关

生成url

```
Dever::url('url');
```

跳转地址

```
Dever::location('url');
```

curl

```
Dever::curl('url');
```

字符串相关

读取正则规则

```
//暂时只有这两种，后续添加更多
Dever::rule('email');
Dever::rule('mobile');
```

id加密

```
Dever::idtostr(1);//加密
Dever::strtoid('ttttt');//解密
```

字符串截取

```
Dever::cut('str', 10, '...');
```

生成随机码

```
Dever::code(4);//形式一，全字母+全数字
Dever::rand(4,0)//形式二，可以选择随机的类型：0全数字，1全字母小写，2全字母大写，3全字母大小写，4数字+字母大小写
```

缓存处理

数据缓存

```
Dever::cache('key', $value, $expire = 300);//当只有key这个参数时，为读取缓存
```

清理header上的缓存时间

```
Dever::clearHeaderCache();
```

模板相关

使用数据绑定形式

```
Dever::view('test');
```

使用传统的smarty形式

```
Dever::render('test', $data);
```

契约规范

基本理念

遵守Dever框架的契约规范，可以更方便的进行开发。在遵守本规范的前提下，请先遵守PSR系列规范。

项目划分

尽量将大项目拆分成若干个小项目。每个项目里有若干个耦合较深的模块。不相干的或者弱相关的公共模块可以独立为一个项目。
如果模块下业务较多，则可以单独建立一个目录，如果是单一业务，请建立一个类文件。
以下的契约规范基本上都是如果书写一个类的说明。

命名空间

类必须加上命名空间。定义为当前项目名\目录名。必须在头部顶部书写

```
<?php
namespace Demo\Src;
```

使用公共类

每个类里最好声明直接使用公共类：

```
use Dever;
```

类名定义

业务级别的程序开发，必须使用面向对象方式编写。
每个项目可以定义很多模块。模块的命名无论是目录还是文件，首字母都必须大写。其余字母小写。

如：

```
//在demo项目下建立src目录，该目录下建立Data.php类文件，内容为：
namespace Demo\Src;
use Dever;
class Data
{}
```

方法名定义

类的方法名首字母为小写，如果是多个单词组成，除首字母外，其余单词的首字母必须为大写。

方法名必须定义为公开还是私有。如果需要定义静态方法，请将static写在public后边。

```
public function getName()
public static function getName()
```

方法体定义

类的方法体必须由大括号包含并且自成一行。

```
public function getName()
{
}
```

方法体行与列

行：最多80个字符。

列：一般不超过30行。

通用组件

基本理念

通用组件是由Dever框架开发的一系列可以高度重用的公共组件。

通用后台

AI自动数据填充系统

过滤审核系统

上传组件

附录

Demo

Demo

访问地址

http://leo.demo.8dev.net/index.php/index

git

git@gitlab.self.com.cn:php/demo.git

注意：域名中的leo可以更换为你自己的主机名。

一些测试

访问首页（后端开发，数据绑定模式）

http://leo.demo.8dev.net/index.php/index

对应的文件：

a、assets/html/index.html：前端页面

b、template/index.php：后端模板解析

c、service/News.php：后端接口

访问某个页面（前端开发，数据绑定模式）

http://leo.demo.8dev.net/index.php/bind

对应的文件：

a、assets/html/bind.html：前端页面

b、service/News.php：后端接口

访问某个页面（仿smarty，mvc模式）

http://leo.demo.8dev.net/index.php/service/news.test_smarty

对应的文件：

a、service/News.php：后端控制器test_smarty_api

b、assets/html/smarty.html：前端页面，需要前后端共同维护开发

直接获取数据

http://leo.demo.8dev.net/index.php/service/news.get

对应的文件：

a、service/News.php：后端控制器get_api

b、database/news.php：后端数据模型

c、database/Model/News.php：后端数据模型方法自定义

测试事务&测试数据更新

http://leo.demo.8dev.net/index.php/service/news.test_commit

对应的文件：

a、service/News.php：后端控制器test_commit_api

b、database/cat.php：后端数据模型&方法自定义

测试自定义事务&测试数据更新

http://leo.demo.8dev.net/index.php/service/news.test_sw_api

对应的文件：

a、service/News.php：后端控制器test_commit_api

b、database/cat.php：后端数据模型&方法自定义

通用后台

http://leo.demo.8dev.net/manage/

请先开通访问权限