文件结构、程序结构及配置

入口文件**index.php**流程

```
//用于调试的配置常量：
define('YII_DEBUG',true);      //启用调试，默认为false
define('YII_TRACE_LEVEL',3);         //出错时错误信息显示的调用堆栈深度，默认为0
//以上选项必须在require yii.php之前定义
//加载Yii框架启动文件
require_once('Yii/framework/yii.php');

$config="protected/config/main.php";//既可以为配置文件路径，也可以为包含配置选项的数组
//即,也可以
$config=(require "protected/config/main.php";);

$app=Yii::createWebApplication($config);//创建WebApplication实例
//$app=new CWebApplication($config);

$app->run(); //运行Controller
```

CApplication构造流程

1. 设置Yii::app()为当前运行实例

2. BasePath及PathAlias设置
3. preinit()（未实现 ）
4. 注册ErrorHandler和ExceptionHandler
5. 注册系统核心组件
6. 应用配置（main.php中的配置将对核心组件起作用）
7. 加载Behaviors（main.php中的配置对此处加载的扩展—— behaviors配置项,不起作用）
8. 加载Components（main.php中的配置对此处加载的组件不起作用）
9. init() ——初始化request组件

   Request组件初始化

   CApplication实例->init() ->getRequest() -> Yii::createComponent('request');
   request组件为系统核心组件
   1. $c=Yii::createComponent(array('class'=>'CHttpRequest')) //new CHttpRequest
   2. $c->init() //Normalize Request,如果开启CSRF校验，则注册beginRequest事件的回调函数$c->**validateCsrfToken**

Run 方法流程

1. 触发onBeginRequest事件,(如果开启CSRF校验，则运行Handler validateCsrfToken)
2. processRequest,运行Controller，流程请参见**Controller**
3. 触发onEndRequest事件，正常结束请求返回

Yii核心参考

Yii extends YiiBase

import($alias,$forceInclude=false)：
$alias为path.to.class形式，将被转换为类文件的路径 path/to/class.php
可以预先使用setPathOfAlias设置path别名，如：
Yii::setPathOfAlias('ext',$this->getBasePath().'/extensions');
import方法只是将此路径添加到include_path中，当真正用到此类时才include这个文件
将$forceInclude设为true可以立即include类文件
$alias解释：
- system：Yii框架命名空间，如import('system.utils.CFormatter')表示Yii/framework/utils/CFormatter.php
- zii：表示Zii目录

以下为CApplication::__construct中设置的alias

- application：一般为配置中所设置的basePath，即应用程序下的protected目录
- webroot：请求的PHP脚本文件所在的目录
- ext：表示application目录下的extensions目录

createComponent($config):
createComponent接受的参数形式

1. Component类名称(或alias,@see import)
2. 含有class键的关联数组，如：
   array(“class”=>”ComponentClassName”,”property1’=>”value1”,”property2”=>”value2”)
   class键表示要创建的组件的名称(同第一条)，其它键将在组件对象创建完成后，赋值为组件对象的属性（即返回的组件对象将具有属性property1，且值为value1等等）
3. $config参数及更多的参数，更多的参数将作为创建组件对象时的构造参数

方法返回被创建的组件对象

trace($msg):

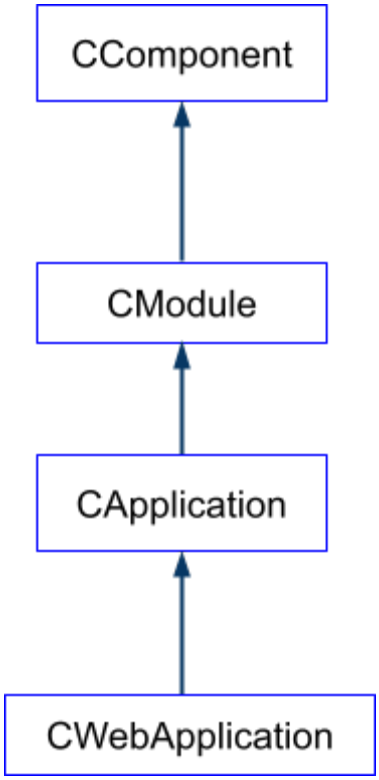
输入LEVEL_TRACE的Log

log($msg,$level=CLogger::LEVEL_INFO,$category='application'):

记录日志，可通过CLogger::getLogs()获得已经记录的日志

app():
返回当前运行的CApplication实例

CWebApplication继承树



Abstract CApplication

CApplication支持四个自定义事件

1. onBeginRequest
2. onEndRequest
3. onException
4. onError

Contructor参数$config

1. basePath,程序根目录
2. 其它属性分别交由CModule::configure方法处理
    a. $config为关联数组，则进行批量属性赋值

CModule

CModule实现功能：

1. Component预加载和管理
2. 模块配置，configure方法对实例属性进行批量赋值
3. 子模块管理

public $preload=array('name1','name2');
预加载的组件

getComponent($id,$createIfNull=true)
根据组件名称获取组件对象，也可以直接通过 $m->componentName获取组件对象

getComponent($config);
$config=array('class'=>'className','property1'=>'value1'...);

Yii::app()->log ;//获取log组件

CComponent

CComponent
CComponent实现了以下三个功能：

1. getter和setter自动化处理
2. 类似于多重继承的Behavior扩展机制
3. 事实注册及调用

getter与setter

```
class ComponentDemo extends CComponent {
        private $_age=12;
        public function getAge() {return $this->_age;}
        public function setAge($value) {$this->_age=$value;}
}
$a=new ComponentDemo();
echo $a->age;//将自动调用getAge方法并返回值（getAge方法不是必须返回值 ）
$a->age=123;//自动调用setAge方法并将设置的值作为其参数
```

Behavior扩展（多重继承）

```
$c=new CComponent();

//$behavior必须实现IBehavior接口
//IBehavior接口是实现attach($component), dettach($component)
//及setEnabled,getEnabled方法(设置是否启用)
//$name用于移除及获取对应behavior

$c->attachBehavior($name,$behavior);
$c->dettachBehavior($name);//移除behavior

//$c将具有$behavior的所有方法及属性($behavior->setEnabled(false)禁用此扩展)

//获取之前绑定的$behavior两种方法
$c->behaviorName;
$c->asa($behaviorName);
```

事件注册与调用

事件属性必须以**on**前缀开头,小写

CComponent的事件机制功能仅为了方便对象实现自定义事件

```
$c=new CComponent();
$c->onclick=create_function('','return;');
$c->attachEventHandler('onclick',array('ClassName','methodName'));//绑定多个事件监听器
$e=new CEvent($c);
$c->raiseEvent('onclick',$e);//触发事件, 逐个调用注册到它上面的监听函数
//在执行中可将$e->handled设为true, 停止继续调用其它的函数
```

属性读取将先尝试**Compoent**公共属性,**setter**与**getter**, 再尝试**Component**的事件属性
最后尝试**$behavior**的属性方法及事件

相关参考:

CEvent extends CComponent

CComponent::raiseEvent($name,$event);第二个参数必须为CEvent类型

$event=new CEvent($sender); //$sender可以为任何对象，之后可通过$event->sender读取
$event->handled=true;/*设置一个bool值，默认false,设为true之后可阻止此事件相关的
其它handler被调用(类似于JS Event的stopPropagation)*/

CEnumerable

枚举类型，没有任何方法及属性，仅用作被所有只包含const的枚举类所继承

CWebApplication配置项

按Yii规约，Web程序配置文件为protected/config/main.php中，测试用配置文件为protected/config/test.php

配置文件必须返回一个关联数组，关联数组$config将作为构造参数传递给CApplication构造函数，$config['basePath']将被设置为命名空间别名"application"的值(即应用程序protected目录)
所有其它的键-值将被批量赋值为CWebApplication的属性-值

CWebApplication继承树中的所有公共属性

CWebApplication

defaultController='site';
默认调用的控制器
默认为SiteControlller，即*protected/controllers/SiteController/php*

$layout='main';
应用程序范围的默认布局
默认为main，即*protected/views/layouts/main.php*

$controllerMap=array();
一个包含"控制器ID=>控制器配置 "的关联数组
当接收到一个请求时，Yii将首先读取此Map
控制器配置为Yii::createComponent所能接受的参数(@see createComponent),但此配置必须为一个Controller的配置（继承自CController）
控制器ID为在URL中显示的路径名

示例:

```
“controllerMap”=>array(
        “goodname”=>”application.controllers.BadnameController”,
        “article”=>array(
                “class”=>”application.controllers.PostController”,
                “layout”=>”article”
        )
)
```

上面的配置（放在main.php配置文件中）将使访问URL */goodname/*时调用BadnameController，而在访问URL */article/*时调用PostController并且将其实例的layout属性设置为article

$catchAllRequest;
设置为一个数组array(“controller/action”)，用于处理所有的请求（用于当需要维护时关闭站点）
数组的第一个元素为一个表示请求控制器的路径字符串，其它的键值对将被设置到$_GET中
示例：

```
“catchAllRequest”=>array(
        'offline/notice',
        'param1'=>'value1',
        'param2'=>'value2',
)
```

所有的请求将被offline控制器的notice动作处理（或其它控制器），所有的请求等同于请求请求URL　offline/notice?param1=value&param2=value2

CApplication

$name='My Application';
没有程序上的意义

$charset='UTF-8';
没有程序上的意义

$sourceLanguage='en_us';
语言配置，关联到所使用的语言包

CModule

$preload=array();
预加载的组件的组件ID（组件的类别名）

预加载的组件将在程序开始时就创建实例

$behaviors=array();
在对象实例创建且配置完成（$config的键值已经被拷贝到对象上时）时自动添加的扩展
即这里列出的**behaviors**不和**CApplication**共享配置
只能在**behaviors**项下进行配置！
（@see CComponent::behavior）

$behaviors值形式为behavior名称与Yii::createComponent所能接受参数值的映射：
array(
'behaviorName'=>array(
'class'=>'BehaviorClassName'
)
)

CWebApplication继承树上实现的setter

CWebApplication
systemViewPath:string 系统使用的*views*文件的路径，默认为*'protected/system/views'*
theme:string 当前使用的主题名称(读取时返回CTheme对象)
homeUrl:string Home Page Url
controller:CController 当前运行的Controller对象
controllerPath:string 包含所有Controller的目录路径，默认为*'protected/controllers'*
viewPath:string 包含所有View文件的目录路径，默认为*'protected/views'*
layoutPath:string 包含layout文件的目录路径，默认为*'protected/views/layouts'*

CApplication

localeDataPath:string
包含系统使用的本地化文件的目录路径，默认为*'framework/i18n/data'*

timeZone:string
时区设置(用作date_default_timezone_set的设置)

language:string
当站点需要支持多语言时，设置此值为语言名称，如zh_CN

extensionPath:string
扩展文件路径，默认为'protected/extensions'

runtimePath:string
runtime文件的路径，默认为*'protected/runtime'*

basePath:string
程序文件默认路径，默认为*'protected'*

id:string
程序的ID，默认为程序路径的CRC32校验值

CModule

params:array
对程序运行没有影响

import:array(setter only)
用于自动import的alias，数组内容为Yii::import(@see import)方法可接受参数的alias字符串

aliases:array(setter only)
alias别名映射，将一个alias(命名空间)映射到一个目录（或已存在的别名）

modules:array
启用的模块
关联数组内容必须为Yii::createComponent(@see createComponent)可接受参数
启用Gii模块的配置Code:

```
'modules'=>array(
        'gii'=>array(
            'class'=>'system.gii.GiiModule',
            'password'=>'123456',
        ),
)
```

components:array
此CApplication需使用的Component，将在创建CApplication实例时预加载（并init）
components值为一包含Yii::createComponent(@see createComponent)能接受参数值的关联数组
获取这些组件的方法，参见CModule::getComponent

CWebApplication继承树上实现的事件

参见CApplication事件

配置**URL**为**Path**模式，去掉**index.php**

```
//启用URLManager Component (main.php array('components'))
'urlManager'=>array(
        'urlFormat'=>'path',
        'rules'=>array(
        '<controller:\w+>/<id:\d+>'=>'<controller>/view',
        '<controller:\w+>/<action:\w+>/<id:\d+>'=>'<controller>/<action>',
        '<controller:\w+>/<action:\w+>'=>'<controller>/<action>',
        ),
        'showScriptName'=>false //去掉index.php
)

//在根目录下添加.htaccess文件，内容为
RewriteEngine On

RewriteCond %{REQUEST_FILENAME} -s [OR]
RewriteCond %{REQUEST_FILENAME} -l [OR]
RewriteCond %{REQUEST_FILENAME} -d

RewriteRule ^.*$ - [NC,L]
RewriteRule ^.*$ index.php [NC,L]
```

Controller与Action的执行

Controller的创建与执行

Controller实例创建流程

1. $app->processRequest,解析获取route(“controllerID/actionID”形式字符串)
    a. 开启catchAllRequest，则使用此此配置(@see catchAllRequest)
    b. UrlManager根据rules,urlFormat等(urlManager配置)解析返回route
2. $app->runController($route),创建Controller实例，初始化:init()，调用:run($actionID)
    a. 构造，注册behaviors(@see Controller::behaviors())
    b. init()方法由用户实现
    c. run($actionID)

Controller run执行流程

1. 创建CAction实例
2. 运行Controller所属的Module（CWebApplication实例）的 beforeControllerAction($controller,$action)方法
3. 应用filters（@see filters()）运行Action
4. 运行Controller所属的Module（CWebApplication实例）的 afterControllerAction($controller,$action)方法

Action的创建与执行

404错误处理

找不到**Controller**

1. UrlManager找不到对应的Controller，则抛出CHttpException 404 异常

2. CApplication::handleException处理异常
3. 触发onException事件，如果Event没有被挂起（handled设为true），继续
4. 查看是否加载了ErrorHandler组件，交由CErrorHandler处理
5. 如果异常没被处理，则直接显示错误信息

ErrorHandler组件配置

```
‘components’=>array(
        ‘errorHandler’=>array(
                ‘errorAction’=>’site/error’   //用来处理404的Action
        )
)
```

找不到**Action**

执行Controller::missingAction方法，此方法默认抛出一个CHttpException 404 异常
可在子类中override missingAction方法

Controller可被覆写的方法

init(): 用于初始化，在实例创建完成，run具体action之前执行
behaviors():array 在构造时给Controller添加扩展，返回包含behavior配置的关联数组 值的形式同配置项::behaviors
filters():array 返回在action执行之前执行的filters列表,REF
missingAction($actionID): 创建Action失败时，则执行missingAction方法
actions(): 当action对应方法找不到时，在此方法返回的ActionMap中查找，REF 自定义Action必须继承自CAction，REF
accessRules(): 返回访问规则数组，REF

Filter

InlineFilter：Controller::filterFilterName形式

InlineFilter配置

```
return array(
    'filterName [+|- action1,action2...]',
)
```

实现InlineFilter的方法签名

```
public function filterFilterName($chain) {}
```

验证通过时执行$chain->run()方法以继续执行下一个filter

CController实现了

a. filterPostOnly方法
b. filterAjaxOnly方法
c. filterAccessControl方法

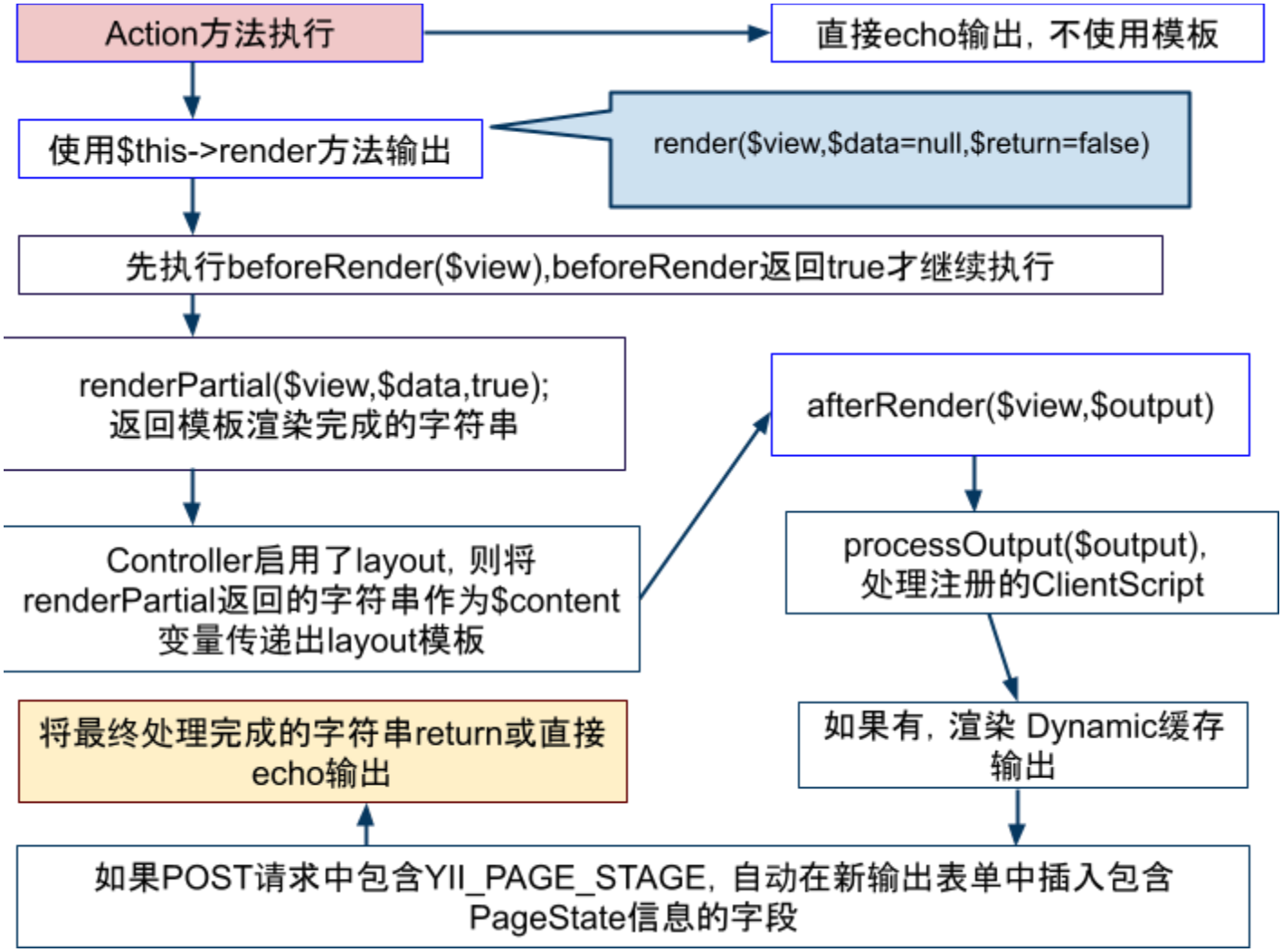
ClassFilter：继承自CFilter的类（@see CFilter）

实现自定义**ClassFilter**

推荐在实现自定义ClassFilter时，采用重写preFilter的方法，而不去重写filter方法

CFilter
**filter($filterChain)**： 执行 filter操作，正常通过时执行$filterChain->run()以继续
init()： 构造完成后运行以初始化
preFilter($filterChain)： 在继承CFilter子类时不去override filter方法，而只覆盖preFilter方法，此方法返回true或false决定filter是否正常通过
**postFilter($filterChain)**： 结合preFilter，只在子类没有覆盖filter方法，只实现preFilter方法时，preFilter正常通过后且执行了$filterChain->run()之后执行

Render输出



Auth Manager & Access Control

User

CWebUser（即核心组件**user**）配置项：

1. allowAutoLogin=false;//是否允许基于cookie的自动登录，如果设为**false**，所有用户数据将保存到**SESSION**中
2. guestName='Guest';//没有登录的用户的name
3. loginUrl=array('site/login');//跳转到的登录页面URL

4. identityCookie;//用于配置用于身份验证Cookie的其它属性，参见CHttpCookie属性,此选项仅在allowAutoLogin设为true可用
5. autoRenewCookie=false;//设置是否每次访问都重新生成用于身份验证的Cookie,此选项仅在allowAutoLogin设为true可用
6. returnUrl;//用户登录后跳转到的URL

CWebUser主要方法

1. **login**(IUserIdentity $identity, integer $duration=0);参见**UserIdentity**
2. **logout**(boolean $destroySession=true);
3. **checkAccess**(string $operation, array $params=array ( ), boolean $allowCaching=true);//参见**Auth**

UserIdentity

实现

1. 继承自**CUserIdentity**
2. 实现**authenticate**方法，通过$this->username和$this->password进行认证，方法返回true表示认证成功

User State

1. 在UserIdentity中重写getPersistenceStates方法，返回需要保存到Cookie中的用户数据（这部分数据不能包含用来认证的数据-比如password）的关联数组
2. 对于登录用户，可以通过Yii::app()->user->stateName获取保存的state值

使用

```
//登录
$identity=new UserIdentity($username,$password);
if ($identity->authenticate()) {
      //Yii::app()->user对象为CWebUser实例
       Yii::app()->user->login($identity,$expires);
}


//权限验证
Yii::app()->user->checkAccess($operation);
```

AuthManager

相关参考
CHttpCookie属性:

1. name
2. value=''
3. domain=''
4. expire=0
5. path='/'
6. secure=false
7. httpOnly=false

Access Control

在**Controller**中配置**Action Access**规则

1. 对需要AccessControl的action设置filters规则
2. 覆写Controller::accessRules方法，返回权限验证规则

Controller中需实现的方法

```
//返回访问权限规则
public function accessRules() {
        return array(
          'allow',  // or 'deny'
          // optional, list of action IDs (case insensitive) that this rule applies to
          'actions'=>array('edit', 'delete'),
          // optional, list of controller IDs (case insensitive) that this rule applies to
          // This option is available since version 1.0.3.
          'controllers'=>array('post', 'admin/user'),
          // optional, list of usernames (case insensitive) that this rule applies to
          // Use * to represent all users, ? guest users, and @ authenticated users
          'users'=>array('thomas', 'kevin'),
          // optional, list of roles (case sensitive!) that this rule applies to.
          'roles'=>array('admin', 'editor'),
          // optional, list of IP address/patterns that this rule applies to
          // e.g. 127.0.0.1, 127.0.0.*
          'ips'=>array('127.0.0.1'),
          // optional, list of request types (case insensitive) that this rule applies to
          'verbs'=>array('GET', 'POST'),
          // optional, a PHP expression whose value indicates whether this rule applies
          // This option is available since version 1.0.3.
          'expression'=>'!$user->isGuest && $user->level==2',
          // optional, the customized error message to be displayed
```

```
            // This option is available since version 1.1.1.
            'message'=>'Access Denied.',
        );
}

//应用AccessControlFilter
public function filters() {
        return array(
                'accessControl',//对所有action应用访问权限控制
                'accessControl + profile',//仅对profile一个action使用
                'accessControl - login,register',//对除login,register以外的action使用
        );
}
```

Logging

配置项

```
return array(
‘components’=>array(
        ‘log’=>array(
                ‘class’=>’CLogRouter’,
                ‘routes’=>array(
                        ‘class’=>’CFileLogRoute’, //使用CWebLogRoute将Log显示是WebPage上
                        ‘levels’=>'error, warning',//为空则记录所有日志
                        )
                )
        )
);
```