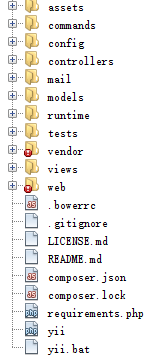
[**Yii2目录结构**](http://www.cnblogs.com/isleep/p/5373144.html)



assets   前端资源文件夹，大致用于管理css js等前端资源文件等

commands   包含命令行命令  文件为控制器文件

config 应用程序的配置文件

controllers 控制器文件夹

mail  就是应用的前后台和命令行的与邮件相关的布局文件等  ——网上查到的，但是我还是不懂是做什么的，以后了解了补充

modles 模型文件夹

runtime 运行时文件，程序运行时生成的临时文件

tests     用于包含各种测试程序——例如用PHPunit生成的测试类文件

vendor  这个目录从字面的意思看，就是各种第三方的程序。 这是Composer安装的其他程序的存放目录，包含Yii框架本身，也放在这个目录下面。 如果你向composer.json 目录            增加了新的需要安装的程序，那么下次调用Composer的时候， 就会把新安装的目录也安装在这个 vendor 下面。

          注：Composer——是 PHP 的一个依赖管理工具。它允许你申明项目所依赖的代码库，它会在你的项目中为你安装他们。

views 视图文件夹

web  入口文件的存放目录，目录从名字可以看出，这是一个对于Web服务器可以访问的目录。 除了这一目录，其他所有的目录不应对Web用户暴露出来。这是安全的需要。

requirements.php  用于测试环境是否符合yii框架的要求

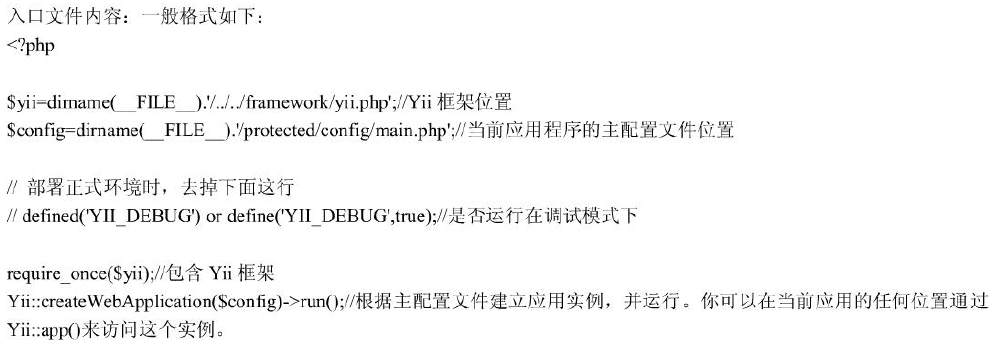
yii 文件

首先，它没有扩展名，我们不好知道其具体类型。 但是从文件内容的第一行 #!/usr/bin/env php 来看，这是一个bash脚本。 第一行在告诉bash，也在告诉我们，这是一个使用PHP运行的脚本。

对于Windows系统而言，命令行的入口脚本仍然是 yii ，但是命令行下无法直接运行。所以， 细心的Yii为我们准备了一个 yii.bat 。这个文件会以 php yii 形式调用PHP来运行入口脚本

**基本概念**

一、入口文件



二、配置文件

应用主体是服务定位器，它部署一组各种不同功能的应用组件来处理请求。在同一个应用中，每个应用组件都有一个独一无二的ID用来区分其他应用组件，可以通过如一下表达式访问应用组件：

\Yii::$app->compinentID

例如,\Yii::$app->db 获取已经注册到应用的DB connection,

\Yii::$app->cache 获取已经注册到应用的primary cache

注：第一次使用以上表达式的时候会创建组件实例，后续再访问会返回此实例，无需要再次创建。

应用组件可以是任意对对象，可以在 应用主体（yii\base\Application）配置compinents属性

如：[

'components' => [

// 使用类名注册 "cache" 组件

'cache' => 'yii\caching\ApcCache',

// 使用配置数组注册 "db" 组件

'db' => [

'class' => 'yii\db\Connection',

'dsn' => 'mysql:host=localhost;dbname=demo',

'username' => 'root',

'password' => '',

],

// 使用函数注册"search" 组件

'search' => function () {

return new app\components\SolrService;

},

],

]

注：1、注册可以使用以上三种方式 。2、系统组件于yii目录下 3、当你配置一个核心组件，不指定它的类名的话就会使用Yii默认指定的类。

1、预载入log组件



2、自动载入模型和组件类



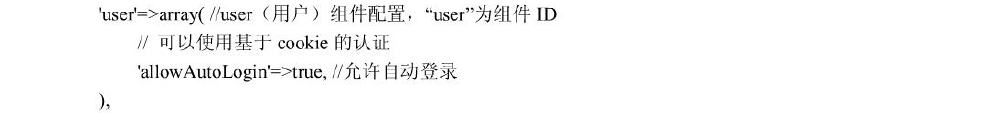
3、设置默认控制器类



4、当前应用的组件配置



4.1、user组件配置



4.2、缓存组件









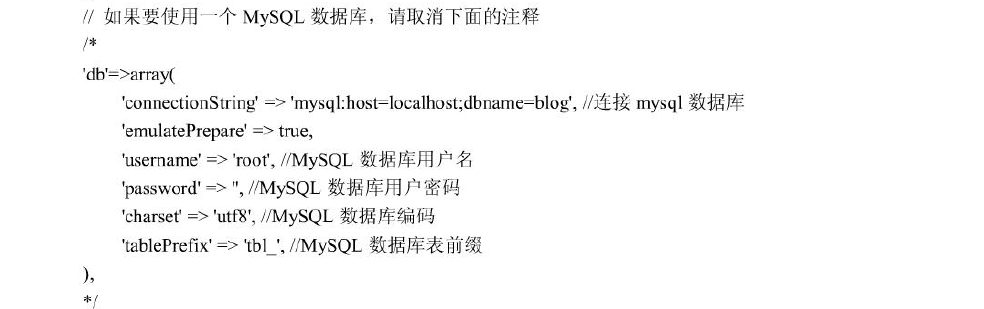




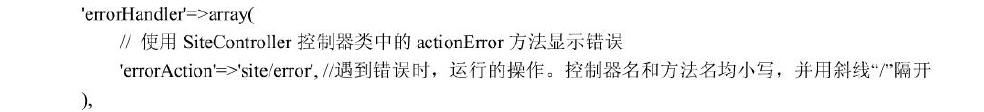


4.3、数据库组件

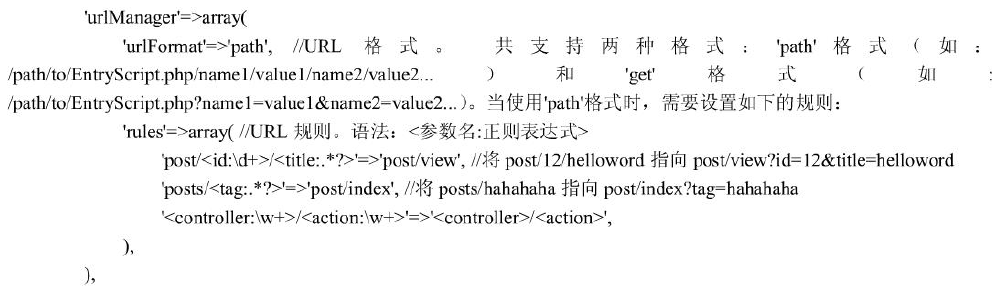




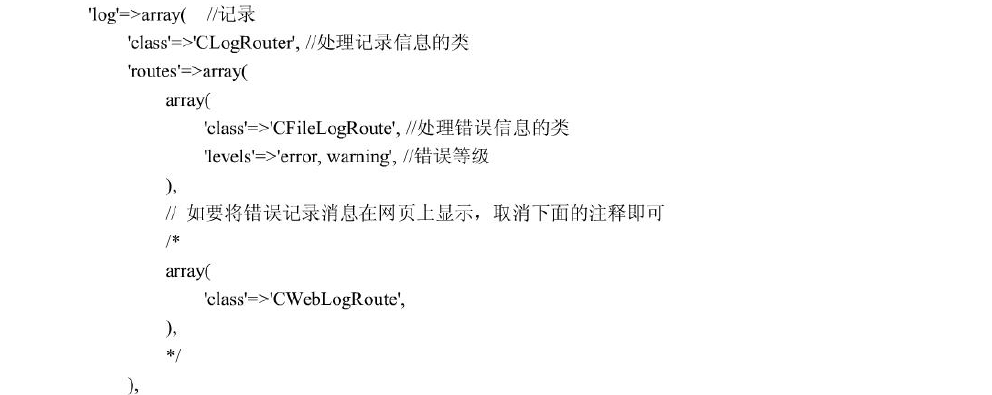
4.4、错误处理



4.5、url路由管理

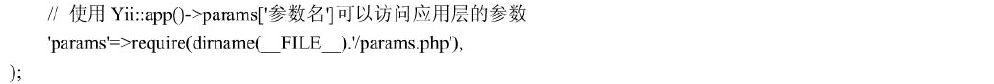


4.6、记录





5、访问应用层参数



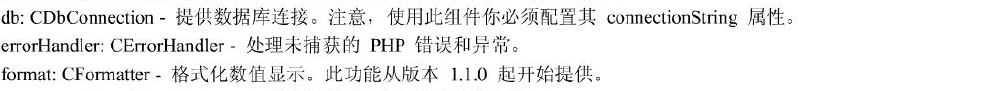
三、核心应用组件

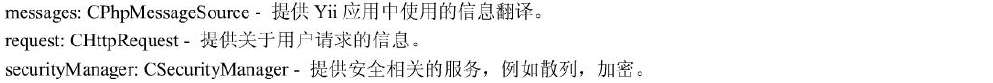
Yii预定义了一系列核心应用组件，提供常见web应用中所用的功能。例如，request组件用于解析用户请求并提供例如URL,cookie等信息。通过配置这些核心组件的属性，我们可以几乎所有的修改Yii的默认行为。

下面我们列出了由CwebApplication预定义的核心组件。









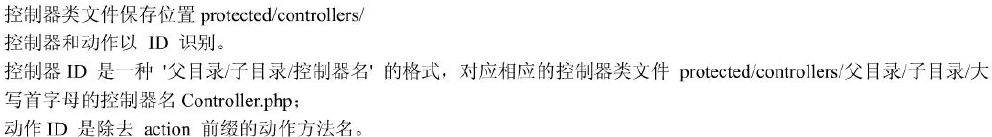






四、控制器（Controller）

控制器是CController类的子类的实例。它在当用户请求时由应用创建 。当一个控制器运行时，它执行所请求的动作（控制器类方法），动作通常会引入所必要的模型并渲染相应的视图。动作，就是一个名字以action开关的控制器类方法。



注：1、可通过配置yii\base\Application::controllerMap来强制上述的控制器ID和类名对应，通常用在使用第三方不能掌控类名的控制器上。如：

[

'controllerMap' => [

// 用类名申明 "account" 控制器

'account' => 'app\controllers\UserController',

// 用配置数组申明 "article" 控制器

'article' => [

'class' => 'app\controllers\PostController',

'enableCsrfValidation' => false,

],

],

]

2、每个应用有一个由yii\base\Application::defaultRoute属性指定的默认控制器。Web applicatiions它的缺省值为‘site’。console applications 控制台应用的缺省值为’help’。可以在应用配置中修改默认控制，如：

[

‘defaultRoute’ => ‘main’,

]

1、路由



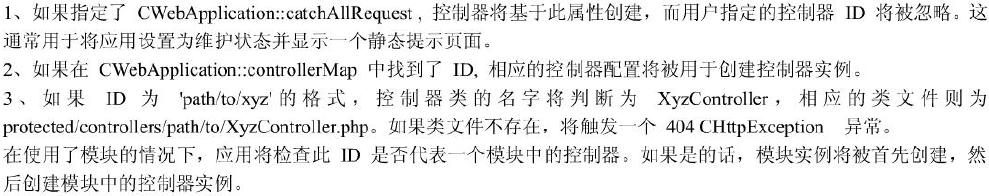


注意：默认情况下，路由是大小写敏感的。可以通过设置 应用 配置中的CUrlManager::caseSensitive为 false使路由对大小写不敏感当在大小写不敏感模式中时，要确保你遵循了相应的规则约定，即：包含控制器类文件的目录名小写且控制器映射和动作映射中使用的键为小写。

路由的格式 ：控制器ID/动作ID或模块ID/控制器ID/动作ID/动作ID(如果是嵌套模块，模块ID就是父模块 ID/子模块ID)。

2、控制器实例化

应用将使用如下规则确定控制器的类以及类文件的位置：



3、动作

动作就是被定义为一个以action单词作为前缀命名的方法。而更高级的方式是定义一个动作类并让控制器在收到请求时将其实例化。这使得动作可以被复用，提高了可复用度。

3.1、定义一个动作类（独立动作）

创建一个独立操作类，需要继承yii\base\Action或它的子类，并实现公有的run()方法。

基本格式如下：

namespace app\components;

use uii\base\Action;

class UpdateAction extends Action

{

public function run(){

//place the action logic here

}

}

3.2使用动作类

为了让控制器注意到这个动作，我们要用如下方式 覆盖控制器类的actions()方法：

class PostController extends Controller

{

public function actions(){

return array(

//使用“应用程序文件夹/ controllers/post/UpdataAction.php”文件中的类业处理edit动作

‘edit’ => ‘application.controllers.post.UpdataAction’,

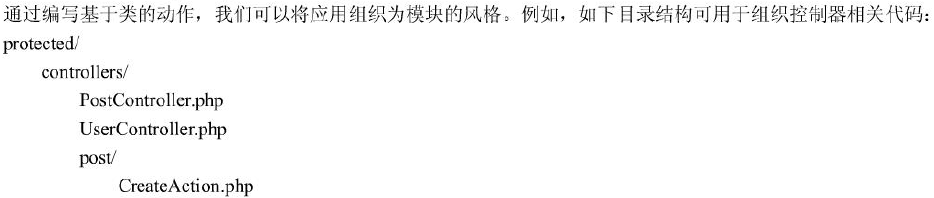
);

}

}

如上所示，我们使用了路径别名‘application.controllers.post.UpdataAction’指定动作类文件为：

“protected/ controllers/post/UpdataAction.php”。











五、过滤器（filter）

过滤器是一段代码，可被 配置在控制器动作执行之前或之后执行。

一个动作可以有多个过滤器。 如有多个过滤器，则按照它们出现在过滤器列表 中的顺序依次执行。过滤器可以阻止动作及后面其他过滤器的执行。

1、基于方法的过滤器

过滤器可以定义为一个控制器类的方法。过滤器方法名必须以filter开头。例如，现有的filterAccessControl方法定义了一个名为accessControl的过滤器。过滤器方法必须为如下结构：

public function filterAccessControll($filterChain){

$filterChain->run(); //调用以继续后续过滤器与动作的执行。

}

$filterChain(过滤器链)是一个CFilterChain的实例，代表与所请求动作相关的过滤器列表。在过滤器方法中，我们可以调用$filterChain->run() 以继续执行后续过滤器和动作。

2、基于对象的过滤器

如动作一样，过滤器也可以是一个对象，它是CFilter或其子类的实例。如下代码定义了一个新的过滤器类：

class PerformanceFilter extends CFilter

{

protected function preFilter($filterChain){

//动作被执行之前应用 的逻辑

return true; //如果动作不应被执行，此处返回 false

}

protected function postFilter($filterChain){

//动作执行之后应用的逻辑

}

}

要对动作应用过滤器，需要覆盖CController::filters()方法。此方法应返回一个过滤器配置数组。例如：

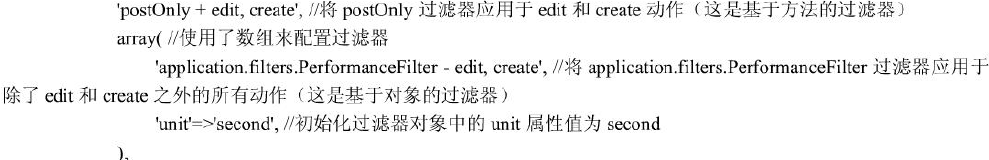
class PostController extends CController

{

public function filters()

{

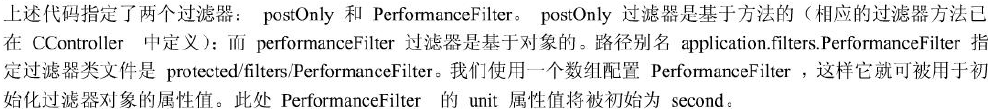
return array(

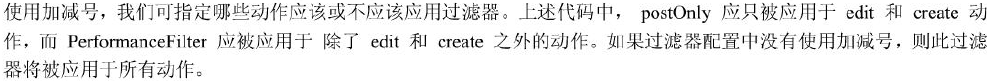


);

}

}





3、使用过滤器

过滤器本质上是一类特殊的 [行为](http://www.yiichina.com/doc/guide/2.0/concept-behaviors)， 所以使用过滤器和 [使用 行为](http://www.yiichina.com/doc/guide/2.0/concept-behaviors#attaching-behaviors)一样。 可以在控制器类中覆盖它的 yii\base\Controller::behaviors() 方法来申明过滤器，如下所示：

**public** **function** **behaviors**()

{

**return** [

[

'class' => 'yii\filters\HttpCache', //过滤器名称

'only' => ['index', 'view'], //应用到这些操作

//'except' => ['index', 'view'], //不应用到这些操作

'lastModified' => **function** ($action, $params) {

$q = **new** \yii\db\Query();

**return** $q->from('user')->max('updated\_at');

},

],

];

}

控制器类的过滤器默认应用到该类的 所有 动作， 你可以配置[only](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$only-detail)属性明确指定控制器应用到哪些动作。 在上述例子中，HttpCache 过滤器只应用到index和view动作。 也可以配置[except](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$except-detail)属性 使一些动作不执行过滤器。

除了控制器外，可在 [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules)或[应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications) 中申明过滤器。 申明之后，过滤器会应用到所属该模块或应用主体的 所有 控制器动作， 除非像上述一样配置过滤器的 [only](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$only-detail) 和 [except](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$except-detail) 属性。

**注意：**在模块或应用主体中申明过滤器，在[only](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$only-detail) 和 [except](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$except-detail) 属性中使用[路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes) 代替动作ID， 因为在模块或应用主体中只用动作ID并不能唯一指定到具体动作。

当一个动作有多个过滤器时，根据以下规则先后执行：

* 预过滤
  + 按顺序执行应用主体中behaviors()列出的过滤器。
  + 按顺序执行模块中behaviors()列出的过滤器。
  + 按顺序执行控制器中behaviors()列出的过滤器。
  + 如果任意过滤器终止动作执行， 后面的过滤器（包括预过滤和后过滤）不再执行。
* 成功通过预过滤后执行动作。
* 后过滤
  + 倒序执行控制器中behaviors()列出的过滤器。
  + 倒序执行模块中behaviors()列出的过滤器。
  + 倒序执行应用主体中behaviors()列出的过滤器。

4、核心过滤器

ii 提供了一组常用过滤器，在 yii\filters 命名空间下， 接下来我们简要介绍这些过滤器。

#### [AccessControl](http://www.yiichina.com/doc/api/2.0/yii-filters-accesscontrol)

AccessControl 提供基于 [rules](http://www.yiichina.com/doc/api/2.0/yii-filters-accesscontrol#$rules-detail) 规则的访问控制。 特别是在动作执行之前，访问控制会检测所有规则 并找到第一个符合上下文的变量（比如用户IP地址、登录状态等等）的规则， 来决定允许还是拒绝请求动作的执行， 如果没有规则符合，访问就会被拒绝。

如下示例表示表示允许已认证用户访问 create 和 update 动作， 拒绝其他用户访问这两个动作。

**use** **yii**\**filters**\**AccessControl**;

**public** **function** **behaviors**()

{

**return** [

'access' => [

'class' => AccessControl::className(),

'only' => ['create', 'update'],

'rules' => [

// 允许认证用户

[

'allow' => **true**,

'roles' => ['@'],

],

// 默认禁止其他用户

],

],

];

}

更多关于访问控制的详情请参阅 [授权](http://www.yiichina.com/doc/guide/2.0/security-authorization) 一节。

#### 认证方法过滤器

认证方法过滤器通过[HTTP Basic Auth](http://en.wikipedia.org/wiki/Basic_access_authentication) 或[OAuth 2](http://oauth.net/2/) 来认证一个用户，认证方法过滤器类在 yii\filters\auth 命名空间下。

如下示例表示可使用[yii\filters\auth\HttpBasicAuth](http://www.yiichina.com/doc/api/2.0/yii-filters-auth-httpbasicauth)来认证一个用户， 它使用基于HTTP基础认证方法的令牌。 注意为了可运行，[user identity class](http://www.yiichina.com/doc/api/2.0/yii-web-user#$identityClass-detail) 类必须 实现 [findIdentityByAccessToken()](http://www.yiichina.com/doc/api/2.0/yii-web-identityinterface#findIdentityByAccessToken()-detail)方法。

**use** **yii**\**filters**\**auth**\**HttpBasicAuth**;

**public** **function** **behaviors**()

{

**return** [

'basicAuth' => [

'class' => HttpBasicAuth::className(),

],

];

}

认证方法过滤器通常在实现RESTful API中使用， 更多关于访问控制的详情请参阅 RESTful [认证](http://www.yiichina.com/doc/guide/2.0/rest-authentication) 一节。

#### [ContentNegotiator](http://www.yiichina.com/doc/api/2.0/yii-filters-contentnegotiator)

ContentNegotiator支持响应内容格式处理和语言处理。 通过检查 GET 参数和 Accept HTTP头部来决定响应内容格式和语言。

如下示例，配置ContentNegotiator支持JSON和XML 响应格式和英语（美国）和德语。

**use** **yii**\**filters**\**ContentNegotiator**;

**use** **yii**\**web**\**Response**;

**public** **function** **behaviors**()

{

**return** [

[

'class' => ContentNegotiator::className(),

'formats' => [

'application/json' => Response::FORMAT\_JSON,

'application/xml' => Response::FORMAT\_XML,

],

'languages' => [

'en-US',

'de',

],

],

];

}

在[应用主体生命周期](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-lifecycle)过程中检测响应格式和语言简单很多， 因此ContentNegotiator设计可被 [引导启动组件](http://www.yiichina.com/doc/guide/2.0/structure-applications#bootstrap)调用的过滤器。 如下例所示可以将它配置在 [应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations)。

**use** **yii**\**filters**\**ContentNegotiator**;

**use** **yii**\**web**\**Response**;

[

'bootstrap' => [

[

'class' => ContentNegotiator::className(),

'formats' => [

'application/json' => Response::FORMAT\_JSON,

'application/xml' => Response::FORMAT\_XML,

],

'languages' => [

'en-US',

'de',

],

],

],

];

**信息：**如果请求中没有检测到内容格式和语言， 使用 formats 和 languages 第一个配置项。

#### [HttpCache](http://www.yiichina.com/doc/api/2.0/yii-filters-httpcache)

HttpCache 利用 Last-Modified 和 Etag HTTP头实现客户端缓存。 例如：

**use** **yii**\**filters**\**HttpCache**;

**public** **function** **behaviors**()

{

**return** [

[

'class' => HttpCache::className(),

'only' => ['index'],

'lastModified' => **function** ($action, $params) {

$q = **new** \yii\db\Query();

**return** $q->from('user')->max('updated\_at');

},

],

];

}

更多关于使用 HttpCache 详情请参阅 [HTTP 缓存](http://www.yiichina.com/doc/guide/2.0/caching-http) 一节。

#### [PageCache](http://www.yiichina.com/doc/api/2.0/yii-filters-pagecache)

PageCache 实现服务器端整个页面的缓存。如下示例所示，PageCache应用在 index 动作， 缓存整个页面 60 秒或 post 表的记录数发生变化。 它也会根据不同应用语言保存不同的页面版本。

**use** **yii**\**filters**\**PageCache**;

**use** **yii**\**caching**\**DbDependency**;

**public** **function** **behaviors**()

{

**return** [

'pageCache' => [

'class' => PageCache::className(),

'only' => ['index'],

'duration' => 60,

'dependency' => [

'class' => DbDependency::className(),

'sql' => 'SELECT COUNT(\*) FROM post',

],

'variations' => [

\Yii::$app->language,

]

],

];

}

更多关于使用PageCache详情请参阅 [页面缓存](http://www.yiichina.com/doc/guide/2.0/caching-page) 一节。

#### [RateLimiter](http://www.yiichina.com/doc/api/2.0/yii-filters-ratelimiter)

RateLimiter 根据 [漏桶算法](http://en.wikipedia.org/wiki/Leaky_bucket) 来实现速率限制。 主要用在实现RESTful APIs，更多关于该过滤器详情请参阅 [Rate Limiting](http://www.yiichina.com/doc/guide/2.0/rest-rate-limiting) 一节。

#### [VerbFilter](http://www.yiichina.com/doc/api/2.0/yii-filters-verbfilter)

VerbFilter检查请求动作的HTTP请求方式是否允许执行， 如果不允许，会抛出HTTP 405异常。 如下示例，VerbFilter指定CRUD动作所允许的请求方式。

**use** **yii**\**filters**\**VerbFilter**;

**public** **function** **behaviors**()

{

**return** [

'verbs' => [

'class' => VerbFilter::className(),

'actions' => [

'index' => ['get'],

'view' => ['get'],

'create' => ['get', 'post'],

'update' => ['get', 'put', 'post'],

'delete' => ['post', 'delete'],

],

],

];

}

#### [Cors](http://www.yiichina.com/doc/api/2.0/yii-filters-cors)

跨域资源共享 [CORS](https://developer.mozilla.org/fr/docs/HTTP/Access_control_CORS) 机制允许一个网页的许多资源（例如字体、JavaScript等） 这些资源可以通过其他域名访问获取。 特别是 JavaScript 的 AJAX 调用可使用 XMLHttpRequest 机制， 由于同源安全策略该跨域请求会被网页浏览器禁止. CORS定义浏览器和服务器交互时哪些跨域请求允许和禁止。

[Cors filter](http://www.yiichina.com/doc/api/2.0/yii-filters-cors) 应在 授权 / 认证 过滤器之前定义， 以保证CORS头部被发送。

**use** **yii**\**filters**\**Cors**;

**use** **yii**\**helpers**\**ArrayHelper**;

**public** **function** **behaviors**()

{

**return** ArrayHelper::merge([

[

'class' => Cors::className(),

],

], **parent**::behaviors());

}

如果要将CORS过滤器添加到你的 API 中的 [yii\rest\ActiveController](http://www.yiichina.com/doc/api/2.0/yii-rest-activecontroller) 类， 还要检查 [REST Controllers](http://www.yiichina.com/doc/guide/2.0/rest-controllers.md%EF%BC%83cors) 中的部分。

CROS过滤器可以通过 [$cors](http://www.yiichina.com/doc/api/2.0/yii-filters-cors#$cors-detail) 属性进行调整。

* cors['Origin']：定义允许来源的数组，可为 ['\*']（任何用户）或 ['http://www.myserver.net', 'http://www.myotherserver.com']。 默认为 ['\*']。
* cors['Access-Control-Request-Method']：允许动作数组如 ['GET', 'OPTIONS', 'HEAD']。默认为 ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'HEAD', 'OPTIONS']。
* cors['Access-Control-Request-Headers']：允许请求头部数组，可为 ['\*'] 所有类型头部 或 ['X-Request-With'] 指定类型头部。默认为 ['\*']。
* cors['Access-Control-Allow-Credentials']：定义当前请求是否使用证书，可为 true, false 或 null (不设置). 默认为 null。
* cors['Access-Control-Max-Age']: 定义请求的有效时间，默认为 86400。

例如，允许来源为 http://www.myserver.net 和方式为 GET，HEAD 和 OPTIONS 的 CORS 如下：

**use** **yii**\**filters**\**Cors**;

**use** **yii**\**helpers**\**ArrayHelper**;

**public** **function** **behaviors**()

{

**return** ArrayHelper::merge([

[

'class' => Cors::className(),

'cors' => [

'Origin' => ['http://www.myserver.net'],

'Access-Control-Request-Method' => ['GET', 'HEAD', 'OPTIONS'],

],

],

], **parent**::behaviors());

}

可以覆盖默认参数为每个动作调整CORS 头部。例如，为 login 动作 增加 Access-Control-Allow-Credentials 参数如下所示：

**use** **yii**\**filters**\**Cors**;

**use** **yii**\**helpers**\**ArrayHelper**;

**public** **function** **behaviors**()

{

**return** ArrayHelper::merge([

[

'class' => Cors::className(),

'cors' => [

'Origin' => ['http://www.myserver.net'],

'Access-Control-Request-Method' => ['GET', 'HEAD', 'OPTIONS'],

],

'actions' => [

'login' => [

'Access-Control-Allow-Credentials' => **true**,

]

]

],

], **parent**::behaviors());

}

六、模型

模型是MVC模式中的一部分，是代表业务数据、规则和逻辑的对象。

可能过继承yii\base\Model或它的子类定义模型类，基类yii\base\Model支持许多实用的特性，如：

属性：代表可像普通类属性或数组一个被访问的业务数据。

属性标签：指定属性显示出来的标签。

块赋值：支持一步给许多属性赋值。

验证规则：确保输入数据符合所申明 的验证规则

数据导出：允许模型数据导出为自定义格式的数组。

1. 访问属性

模型通过 属性 来代表业务数据，每个属性像是模型的公有可访问属性， [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes()-detail) 指定模型所拥有的属性。

可像访问一个对象属性一样访问模型的属性:

$model = **new** \app\models\ContactForm;

// "name" 是ContactForm模型的属性

$model->name = 'example';

**echo** $model->name;

也可像访问数组单元项一样访问属性，这要感谢[yii\base\Model](http://www.yiichina.com/doc/api/2.0/yii-base-model)支持 [ArrayAccess 数组访问](http://php.net/manual/en/class.arrayaccess.php) 和 [ArrayIterator 数组迭代器](http://php.net/manual/en/class.arrayiterator.php):

$model = **new** \app\models\ContactForm;

// 像访问数组单元项一样访问属性

$model['name'] = 'example';

**echo** $model['name'];

// 迭代器遍历模型

**foreach** ($model **as** $name => $value) {

**echo** "$name: $value\n";

}

1. 属性标签

当属性显示或获取输入时，经常要显示属性相关标签， 例如假定一个属性名为firstName， 在某些地方如表单输入或错误信息处，你可能想显示对终端用户来说更友好的 First Name 标签。

可以调用 [yii\base\Model::getAttributeLabel()](http://www.yiichina.com/doc/api/2.0/yii-base-model#getAttributeLabel()-detail) 获取属性的标签，例如：

$model = **new** \app\models\ContactForm;

// 显示为 "Name"

**echo** $model->getAttributeLabel('name');

默认情况下，属性标签通过[yii\base\Model::generateAttributeLabel()](http://www.yiichina.com/doc/api/2.0/yii-base-model#generateAttributeLabel()-detail)方法自动从属性名生成. 它会自动将驼峰式大小写变量名转换为多个首字母大写的单词， 例如 username 转换为 Username， firstName 转换为 First Name。

如果你不想用自动生成的标签，可以覆盖 [yii\base\Model::attributeLabels()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributeLabels()-detail) 方法明确指定属性标签， 例如：

**namespace** **app**\**models**;

**use** **yii**\**base**\**Model**;

**class** **ContactForm** **extends** **Model**

{

**public** $name;

**public** $email;

**public** $subject;

**public** $body;

**public** **function** **attributeLabels**()

{

**return** [

'name' => 'Your name',

'email' => 'Your email address',

'subject' => 'Subject',

'body' => 'Content',

];

}

}

应用支持多语言的情况下，可翻译属性标签， 可在 [attributeLabels()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributeLabels()-detail) 方法中定义，如下所示:

**public** **function** **attributeLabels**()

{

**return** [

'name' => \Yii::t('app', 'Your name'),

'email' => \Yii::t('app', 'Your email address'),

'subject' => \Yii::t('app', 'Subject'),

'body' => \Yii::t('app', 'Content'),

];

}

甚至可以根据条件定义标签，例如通过使用模型的 [scenario场景](http://www.yiichina.com/doc/guide/2.0/structure-models#scenarios)， 可对相同的属性返回不同的标签。

**信息：**属性标签是 [视图](http://www.yiichina.com/doc/guide/2.0/structure-views)一部分， 但是在模型中申明标签通常非常方便，并可行程非常简洁可重用代码。

3、场景

模型可能在多个 场景 下使用，例如 User 模块可能会在收集用户登录输入， 也可能会在用户注册时使用。在不同的场景下， 模型可能会使用不同的业务规则和逻辑， 例如 email 属性在注册时强制要求有，但在登陆时不需要。

模型使用 yii\base\Model::scenario 属性保持使用场景的跟踪， 默认情况下，模型支持一个名为 default 的场景， 如下展示两种设置场景的方法:

// 场景作为属性来设置

$model = **new** User;

$model->scenario = 'login';

// 场景通过构造初始化配置来设置

$model = **new** User(['scenario' => 'login']);

默认情况下，模型支持的场景由模型中申明的 [验证规则](http://www.yiichina.com/doc/guide/2.0/structure-models#validation-rules) 来决定， 但你可以通过覆盖[yii\base\Model::scenarios()](http://www.yiichina.com/doc/api/2.0/yii-base-model#scenarios()-detail)方法来自定义行为， 如下所示：

**namespace** **app**\**models**;

**use** **yii**\**db**\**ActiveRecord**;

**class** **User** **extends** **ActiveRecord**

{

**const** SCENARIO\_LOGIN = 'login';

**const** SCENARIO\_REGISTER = 'register';

**public** **function** **scenarios**()

{

**return** [

**self**::SCENARIO\_LOGIN => ['username', 'password'],

**self**::SCENARIO\_REGISTER => ['username', 'email', 'password'],

];

}

}

**信息：**在上述和下述的例子中，模型类都是继承[yii\db\ActiveRecord](http://www.yiichina.com/doc/api/2.0/yii-db-activerecord)， 因为多场景的使用通常发生在[Active Record](http://www.yiichina.com/doc/guide/2.0/db-active-record) 类中.

scenarios() 方法返回一个数组，数组的键为场景名，值为对应的 active attributes活动属性。 活动属性可被 [块赋值](http://www.yiichina.com/doc/guide/2.0/structure-models#massive-assignment) 并遵循[验证规则](http://www.yiichina.com/doc/guide/2.0/structure-models#validation-rules) 在上述例子中，username 和 password 在login场景中启用，在 register 场景中, 除了 username and password 外 email 也被启用。

scenarios() 方法默认实现会返回所有[yii\base\Model::rules()](http://www.yiichina.com/doc/api/2.0/yii-base-model#rules()-detail)方法申明的验证规则中的场景， 当覆盖scenarios()时，如果你想在默认场景外使用新场景， 可以编写类似如下代码：

**namespace** **app**\**models**;

**use** **yii**\**db**\**ActiveRecord**;

**class** **User** **extends** **ActiveRecord**

{

**const** SCENARIO\_LOGIN = 'login';

**const** SCENARIO\_REGISTER = 'register';

**public** **function** **scenarios**()

{

$scenarios = **parent**::scenarios();

$scenarios[**self**::SCENARIO\_LOGIN] = ['username', 'password'];

$scenarios[**self**::SCENARIO\_REGISTER] = ['username', 'email', 'password'];

**return** $scenarios;

}

}

场景特性主要在[验证](http://www.yiichina.com/doc/guide/2.0/structure-models#validation-rules) 和 [属性块赋值](http://www.yiichina.com/doc/guide/2.0/structure-models#massive-assignment) 中使用。 你也可以用于其他目的， 例如可基于不同的场景定义不同的 [属性标签](http://www.yiichina.com/doc/guide/2.0/structure-models#attribute-labels)

4、验证规则

当模型接收到终端用户输入的数据， 数据应当满足某种规则(称为 验证规则, 也称为 业务规则)。 例如假定ContactForm模型，你可能想确保所有属性不为空且 email 属性包含一个有效的邮箱地址， 如果某个属性的值不满足对应的业务规则， 相应的错误信息应显示，以帮助用户修正错误。

可调用 [yii\base\Model::validate()](http://www.yiichina.com/doc/api/2.0/yii-base-model#validate()-detail) 来验证接收到的数据， 该方法使用[yii\base\Model::rules()](http://www.yiichina.com/doc/api/2.0/yii-base-model#rules()-detail)申明的验证规则来验证每个相关属性， 如果没有找到错误，会返回 true， 否则它会将错误保存在 yii\base\Model::errors 属性中并返回false，例如：

$model = **new** \app\models\ContactForm;

// 用户输入数据赋值到模型属性

$model->attributes = \Yii::$app->request->post('ContactForm');

**if** ($model->validate()) {

// 所有输入数据都有效 all inputs are valid

} **else** {

// 验证失败：$errors 是一个包含错误信息的数组

$errors = $model->errors;

}

通过覆盖 [yii\base\Model::rules()](http://www.yiichina.com/doc/api/2.0/yii-base-model#rules()-detail) 方法指定模型 属性应该满足的规则来申明模型相关验证规则。 下述例子显示ContactForm模型申明的验证规则:

**public** **function** **rules**()

{

**return** [

// name, email, subject 和 body 属性必须有值

[['name', 'email', 'subject', 'body'], 'required'],

// email 属性必须是一个有效的电子邮箱地址

['email', 'email'],

];

}

一条规则可用来验证一个或多个属性，一个属性可对应一条或多条规则。 更多关于如何申明验证规则的详情请参考[验证输入](http://www.yiichina.com/doc/guide/2.0/input-validation) 一节.

有时你想一条规则只在某个 [场景](http://www.yiichina.com/doc/guide/2.0/structure-models#scenarios) 下应用，为此你可以指定规则的 on 属性， 如下所示:

**public** **function** **rules**()

{

**return** [

// 在"register" 场景下 username, email 和 password 必须有值

[['username', 'email', 'password'], 'required', 'on' => 'register'],

// 在 "login" 场景下 username 和 password 必须有值

[['username', 'password'], 'required', 'on' => 'login'],

];

}

如果没有指定 on 属性，规则会在所有场景下应用， 在当前yii\base\Model::scenario 下应用的规则称之为 active rule活动规则。

一个属性只会属于scenarios()中定义的活动属性且在rules() 申明对应一条或多条活动规则的情况下被验证。

6、安全属性

块赋值只应用在模型当前yii\base\Model::scenario 场景[yii\base\Model::scenarios()](http://www.yiichina.com/doc/api/2.0/yii-base-model#scenarios()-detail)方法 列出的称之为 安全属性 的属性上，例如，如果User模型申明以下场景， 当当前场景为login时候，只有username and password 可被块赋值， 其他属性不会被赋值。

**public** **function** **scenarios**()

{

**return** [

'login' => ['username', 'password'],

'register' => ['username', 'email', 'password'],

];

}

**信息：**块赋值只应用在安全属性上， 因为你想控制哪些属性会被终端用户输入数据所修改， 例如，如果 User 模型有一个permission属性对应用户的权限， 你可能只想让这个属性在后台界面被管理员修改。

由于默认[yii\base\Model::scenarios()](http://www.yiichina.com/doc/api/2.0/yii-base-model#scenarios()-detail)的实现会返回 [yii\base\Model::rules()](http://www.yiichina.com/doc/api/2.0/yii-base-model#rules()-detail)所有属性和数据， 如果不覆盖这个方法，表示所有只要出现在活动验证规则中的属性都是安全的。

为此，提供一个特别的别名为 safe 的验证器来申明 哪些属性是安全的不需要被验证， 如下示例的规则申明 title 和 description 都为安全属性。

**public** **function** **rules**()

{

**return** [

[['title', 'description'], 'safe'],

];

}

7、非安全属性

如上所述，[yii\base\Model::scenarios()](http://www.yiichina.com/doc/api/2.0/yii-base-model#scenarios()-detail) 方法提供两个用处：定义哪些属性应被验证，定义哪些属性安全。 在某些情况下，你可能想验证一个属性但不想让他是安全的， 可在scenarios()方法中属性名加一个惊叹号 !。 例如像如下的secret属性。

**public** **function** **scenarios**()

{

**return** [

'login' => ['username', 'password', '!secret'],

];

}

当模型在 login 场景下，三个属性都会被验证， 但只有 username和 password 属性会被块赋值， 要对secret属性赋值，必须像如下例子明确对它赋值。

$model->secret = $secret;

The same can be done in rules() method:

**public** **function** **rules**()

{

**return** [

[['username', 'password', '!secret'], 'required', 'on' => 'login']

];

}

In this case attributes username, password and secret are required, but secret must be assigned explicitly.

8、导出数据

模型通常要导出成不同格式，例如，你可能想将模型的一个集合转成JSON或Excel格式， 导出过程可分解为两个步骤：

* 模型转换成数组；
* 数组转换成所需要的格式。

你只需要关注第一步，因为第二步可被通用的 数据转换器如[yii\web\JsonResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-jsonresponseformatter)来完成。

将模型转换为数组最简单的方式是使用 [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes()-detail) 属性， 例如：

$post = \app\models\Post::findOne(100);

$array = $post->attributes;

[yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes()-detail) 属性会返回 所有 [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes()-detail) 申明的属性的值。

更灵活和强大的将模型转换为数组的方式是使用 yii\base\Model::toArray() 方法， 它的行为默认和 [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes()-detail) 相同， 但是它允许你选择哪些称之为字段的数据项放入到结果数组中并同时被格式化。 实际上，它是导出模型到 RESTful 网页服务开发的默认方法， 详情请参阅[响应格式](http://www.yiichina.com/doc/guide/2.0/rest-response-formatting).

9、字段

字段是模型通过调用yii\base\Model::toArray() 生成的数组的单元名。

默认情况下，字段名对应属性名，但是你可以通过覆盖 [fields()](http://www.yiichina.com/doc/api/2.0/yii-base-model#fields()-detail) 和/或 yii\base\Model::extraFields() 方法来改变这种行为， 两个方法都返回一个字段定义列表，fields() 方法定义的字段是默认字段， 表示toArray()方法默认会返回这些字段。 extraFields()方法定义额外可用字段， 通过toArray()方法指定$expand参数来返回这些额外可用字段。 例如如下代码会返回fields()方法定义的所有字段和extraFields()方法定义的prettyName and fullAddress字段。

$array = $model->toArray([], ['prettyName', 'fullAddress']);

可通过覆盖 fields() 来增加、删除、重命名和重定义字段， fields() 方法返回值应为数组， 数组的键为字段名，数组的值为对应的可为属性名或匿名函数返回的字段定义对应的值。 特使情况下，如果字段名和属性定义名相同，可以省略数组键， 例如：

// 明确列出每个字段，特别用于你想确保数据表或模型

// 属性改变不会导致你的字段改变(保证后端的API兼容)。

**public** **function** **fields**()

{

**return** [

// 字段名和属性名相同

'id',

// 字段名为 "email"，对应属性名为 "email\_address"

'email' => 'email\_address',

// 字段名为 "name", 值通过PHP代码返回

'name' => **function** () {

**return** $this->first\_name . ' ' . $this->last\_name;

},

];

}

// 过滤掉一些字段，特别用于

// 你想继承父类实现并不想用一些敏感字段

**public** **function** **fields**()

{

$fields = **parent**::fields();

// 去掉一些包含敏感信息的字段

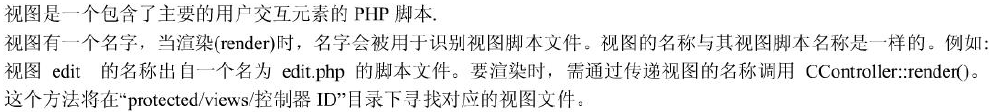
**unset**($fields['auth\_key'], $fields['password\_hash'], $fields['password\_reset\_token']);

**return** $fields;

}

**警告：**由于模型的所有属性会被包含在导出数组，最好检查数据确保没包含敏感数据， 如果有敏感数据，应覆盖 fields() 方法过滤掉， 在上述列子中，我们选择过滤掉 auth\_key, password\_hash and password\_reset\_token。

七、视图











1、布局

布局是一种用来修饰视图的特殊的视图文件。它通常包含了用户界面中通用 的一部份视图。例如：布局可以包含header和footer的部分，然后把内容嵌入其间。

......header here ......

<?=$cotent;?>

......footer here......

其中的$content储存了内容视图的渲染结果。

当使用render()时，布局被隐式应用。视图脚本$app/views/layouts/main.php是默认的布局文件。可配置yii\base\Module::layoutPath来自定义应用或模块的布局默认路径。要渲染一个不带布局的视图，则需要调用 renderPartial()。

2、小物件

注：小物件与小部件代表同一个事物。

2.1、创建小部件：

Exp1:

新建HelloWidget.php文件

|  |
| --- |
| <?php  Namespace app\components;  Use yii\base\Widget;  Class Hellowidget extends Widget{ //继承后具有小部件的属性  Public $catId = 0;  Public function init(){ //此方法会被CWidget::begin()调用  Parent::init();  }  Public function run(){ //此方法会被CWidget::end()调用  Parent::run();  Echo $this->catId;  }  }  ?> |

注：1、原则上一个小部件只要有run()函数，这个小部件就可以运行，但通常会用到init()初始化。

2、小部件中的视图存放在componets\views目录

3、在小部件中可以使用视图。如：

|  |
| --- |
| Public function run(){  Parent::run();  Return $this->render(view:’hello’);  } |

2.2、使用小部件：

方式一：

Exp2:在视图中使用小部件

|  |
| --- |
| <?php  Use app\components\HelloWidget;  ?>  <?= HelloWidget::widge();?> <!—获取小部件运行后输出的数据 -- > |

方式二：(如果所传的参数很多，可以使用以下方式)

|  |
| --- |
| <?= HelloWidget::widge([‘catId’=>9]); ?> <! – 小部件获取视图数据的方式 -- > |

2.3、小部件与缓冲

Exp3\_1:

|  |
| --- |
| <?php  Use app\components\HelloWidget;  ?>  <?php HelloWidget::begin();?>  <div></div>  <?php HelloWidget::end();?> |

Exp3\_2:(获取缓冲区的所有内容)

|  |
| --- |
| <?php  namespace app\components;  Use yii\base\Widget;  Class HelloWidget extends Widget{  Public $catId = 0;  Public function init(){  Parent::init();  Ob\_start(); //开启缓冲  }  Public function run(){  Parent::run();  $content = ob\_get\_clean(); //得到缓冲区的所有内容，然后清除缓冲区  Return $this-> render(view:’hello’,[ //把内容传给小部件视图  ‘content’ => $content  ]);  }  } |

Exp3\_3:小部件视图输出传来的数据，与控制器视图输出数据的方法一致

|  |
| --- |
| <h1><?= $content;?> </h1> |

2.4、小部件的beforeRun() 和 afterRun();

在run()执行之前需要执行一些事情，可以启用beforRun()；afterRun()同理。

注：当beforeRun();返回false。后面的run()就不再执行。如：

|  |
| --- |
| Public function beforeRun(){ //重载beforeRun()函数  Parent::beforeRun();  //Return true;  Return false;  } |

|  |
| --- |
| 总结：小物件可以 像下件控制器一样拥有它自己的视图。  默认情况下，小物件的视图文件位于包含了小物件类文件目录的views子目录之下（protected\componets\views）。这些视图可以通过调用CWidget::render()渲染，这一点和控制器很相似。唯一不同的是小物件的视图没有布局文件支持。另处小物件视图中的$this指向小物件实例而不是控制器实例。 |

2.5、小部件完整示例：

|  |
| --- |
|  |

3、系统视图

系统视图的渲染通常用于展示Yii的错误和日志信息。

系统视图的命名遵从了一些规则。比如像“errorXXX”这样的名称就是用于渲染展示错误号XXX的CHttpException的视图。例如，如果CHttpException抛出一个404错误，那么error404就会被显示。

在yii2\views下，Yii提供了一系列默认的系统视图，他们可以通过在protected\views\system下创建同名视图文件进行自定义。

八组件

Yii应用建立于组件之上。组件是CComponent或其子类的实例。使用组件主要涉及访问的属性以及触发或处理它的时间。基类CComponent指定了如何定义属性和事件。

1、组件属性

组件的属性就像对象的公共成员变量。它是可读写的。

要定义一个组件属性，我们只需要在组件类中定义一个公共成员变量即可。

更灵活的方式是定义其getter和setter方法，例如:

public function getTextWidth(){ //获取textWidth属性

return $this->\_textWidth;

}

public function setTextWidth($value){ //设置textWidth属性

$this->\_textWidth = $value;

}

上述代码定义了一个可写的属性名为textWidth(名字大小写不敏感)成员属性。使用getTextWidth()读取属性。或使用setTextWidth()写入属性。如果setter方法没有定义，则属性将是只读的，如是对其写入则会抛出一个异常。使用getter和setter方法定义一个属性有一个好处：即当读取 或写入属性时，可以执行额外的逻辑（例如，执行验证，触发事件）。

注意：通过getter/setter定义的属性和类成员变量之间有一个细微的差异：属性的名字是大小写不敏感的，而类成员变量是大小写敏感的。

2、组件事件

组件事件是一些特殊的属性，它们使用一些称作事件句柄（event handlers）的方法作为其值 。分配一个方法到一个事件将会引起方法在事件被唤起处自动被调用。因此，一个组件的行为可能会被一种在部件开发过程中不可预见的方式修改。

组件事件以on开头的命名方式定义。和属性通过getter/setter方法来定义 的命名方式一样，事件的名称是大小写不敏感的。以下代码定义了一个onClicked事件：

public function onClicked($event)

{

$this->raiseEvent(‘onClicked’,$event);

}

这里作为事件参数的$event是CEvent或其子类实例。

我们可以分配一个方法到此事件，如下所示：

$component -> onClicked = $callback;

这里的$callback 指向了一个有效的php回调。它可以 是一个全局函数也可以 是类中的一个方法 。如果是后者，它必须以一个数组 的方式担任 ：array($object,’methodName’)。

事件句柄的结构如下：

function 方法名($event) {...}

这里的$event{...}

即描述事件的参数（它来源于raiseEvent()调用）。$event参数是CEvent或其子类的实例。至少，它包含了关于谁触发了些事件的信息。

事件句柄也可以是一个php5.3以后支持的匿名函数。例如：

$component -> onClicked = function($event){}

如果我们现在调用onClicked()，onClicked事件将被 触发（在onClicked()中），附属的事件句柄将被自动调用。

一个事件可以绑定多个句柄。当事件触发时，这些句柄将被按照它们绑定到事件的顺序依次执行。如果句柄决定组织后续句柄被执行，它会设置$event -> handled为true。

3、组件行为

组件已添加了对mixin的支持，并可以绑定定个或多个行为。行为是个对象，其力方法可以披它绑定的部件通过收

集功能的方式来实现继承(inherited).一个部件可以以‘多重继承’的方法实现多个行为的绑定。

行为类必须实现Ibehavior接口。

大多数行为可以自CAcitveRecordBehavior。如果一行为需要绑定到一个模型，它也可以从专为模型实现绑定特性的CModelBehavior或CActive$ecordBehavior继承。

要使用一个行为，它必须先通过调用此行为的attach()方法绑定到一个组件。然后我们就可以通过组件调用此行为方法：

//$name在组件中实现了对行为的唯一识别

$component -> attachBehavior($name,$behavior);

//test()是行为中的方法。

$componnent -> test();

已经绑定的行为 可以像一个组件中的精通属性一样访问。例如，如果一个名为tree行为绑定到了一个组件，我们就可以通过如下代码获得指向此行为的引用。

$behavior = $component -> tree;

//等于下行代码：

$behavior = $component -> asa(‘tree’);

行为是可以被临时朾的， 此时它的方法就会在组件中失效。例如：

$component -> disableBehavior($name);

//下面的方法将抛出一个异常

$compinent -> test();

//启用指定行为

$component -> enableBehavior($name);

//现在就可以 使用了

$component -> test();

注：1、两个同名行为绑定到同一个组件下是有可能 的。在这种情况下，先绑定的行为则拥有优先权。

2、当Behavior和events，一起使用时，行为会更加强大，当行为被绑定到组件时，行为里的一些方法就可以绑定到组件的一些事件上了。这样一来，行为就有机观察或者改变组件的常规执行流程。

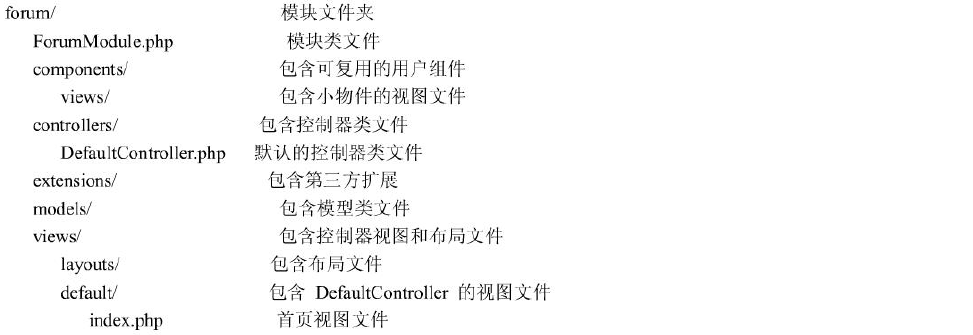
3、一个行为的属性也可以通过绑定到的组件来访问。这些属性包含公共成员变量以及通过getter和/或setter方式设置的属性。例如，若一个行为有一个xyz的属性，此行为被绑定到组件$a，然后我们可以使用表达式$a -> xyz访问此行为的属性。

九、Yi2i中的模块

模块是独立的软件单元，由模型，视图，控制器和其他支持组件组成。终端用户可以访问在应用主体中已安装的模块的控制器，模块被当成小应用主体来看待，和应用主体不同的是，模块不能单独部署，必须属性某个应用主体。

1. 创建模块

模块组织在一个目录中，目录名即为模块的唯一ID。模块目录的结构跟应用基础目录很相似。下面列出了一个fourm的模块的典型目录结构：



注：模块必须是继承自yii\base\Module的模块类。类的名字通过表达式ucfirst($id).’Module’确定，其中的$id代表模块 的ID(或者说模块的目录名)。模块类是存储模块代码间可共享信息的中心位置。例如 ，我们可以使用yii\Web\Module::params存储模块参数，使用yii\Web\Module::components分享模块极的应用组件。

模块类大致定义示例：

**namespace** **app**\**modules**\**forum**;

**class** **Module** **extends** \**yii**\**base**\**Module //创建模块的必要条件**

{

**public** **function** **init**() **//新建模块的必要方法**

{

**parent**::init(); **//必须初始化父类模块**

$this->params['foo'] = 'bar'; **//配置少量初始化属性**

// ... 其他初始化代码 ...

}

}

配置大量初始化模块属性， 可将他们保存在[配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations) 并在init()中使用以下代码加载：

**public** **function** **init**()

{

**parent**::init();

// 从config.php加载配置来初始化模块

\Yii::configure($this, **require**(**\_\_DIR\_\_** . '/config.php'));

}

config.php配置文件可能包含以下内容，类似 [应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations).

<?php

**return** [

'components' => [

// list of component configurations

],

'params' => [

// list of parameters

],

];

1. 使用模块

要使用模块，首先将模块目录放在应用基础目录的modules文件夹中。然后在应用的modules属性中声明模块ID。例如，为了使用上面的forum模块，我们可以使用如下应用配置：

$confing = [

‘modules’ => [

‘forum=> [

‘class’ => ‘app\modules\forum\Module’

]

]

]

模块也可以在配置时带有初始属性值。做法和配置应用组件很类似。例如，forum模块可以在其模块类中有一个名为postPerPage的属性，它可以在应用配置中配置如下：

$confing = [

‘modules’ => [

‘forum=> [

‘class’ => ‘app\modules\forum\Module’,

‘postPerPage’ => ‘20’

]

}

]

提示：

1. 访问模块实例

模块的实例可以通过当前活动控制器的module属性访问。在模块实例中，我们可以访问在模块级中共享的信息。例如：为访问上面的postPerPage信息，我们可使用如下表达式：

$postPerPage = Yii::$app -> controller -> module -> postPerPage;

//如$this引用的是控制器实例，则可以使用下行语句

$postPerPage = $this -> module -> postPerPage;

1. 访问模块中控制器动作

模块中的控制器动作可以通过路由“模块ID/控制器ID/动作ID“访问。此路由对应的url即<http://www.example.com/index.php?r> =模块ID/控制器ID/动作ID。

3、模块嵌套

模块可以无限级嵌套。子模块必须定义在其父模块的modules属性中，就像前面在应用配置中定义模块一样。要访问子模块中的控制器动作路由如：父模块ID/子模块ID/控制器ID/动作ID。 例如：

**namespace** **app**\**modules**\**forum**;

**class** **Module** **extends** \**yii**\**base**\**Module**

{

**public** **function** **init**()

{

**parent**::init();

$this->modules = [

'admin' => [

// 此处应考虑使用一个更短的命名空间

'class' => 'app\modules\forum\modules\admin\Module',

],

];

}

}

在嵌套模块中的控制器，它的路由应包含它所有祖先模块的ID， 例如forum/admin/dashboard/index 代表 在模块forum中子模块admin中dashboard控制器的index操作。

**信息：**[getModule()](http://www.yiichina.com/doc/api/2.0/yii-base-module#getModule()-detail) 方法只返回子模块的直属的 父模块。 [yii\base\Application::$loadedModules](http://www.yiichina.com/doc/api/2.0/yii-base-application#$loadedModules-detail) 保存了已加所有载模块的属性, 包括两者的子模块和 嵌套模块，并用他们的类名进行索引。

4、模块中的控制器

创建模块的控制器时，惯例是将控制器类放在模块类命名空间的controllers子命名空间中， 也意味着要将控制器类文件放在模块 yii\base\Module::basePath目录中的controllers子目录中。 例如，上小节中要在forum模块中创建post控制器， 应像如下申明控制器类：

**namespace** **app**\**modules**\**forum**\**controllers**;

**use** **yii**\**web**\**Controller**;

**class** **PostController** **extends** **Controller**

{

// ...

}

可配置[yii\base\Module::$controllerNamespace](http://www.yiichina.com/doc/api/2.0/yii-base-module#$controllerNamespace-detail)属性来自定义控制器类的命名空间， 如果一些控制器不再该命名空间下，可配置[yii\base\Module::$controllerMap](http://www.yiichina.com/doc/api/2.0/yii-base-module#$controllerMap-detail)属性让它们能被访问， 这类似于 [应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#controller-map) 所做的。

5、模块中的视图

视图应放在模块的yii\base\Module::basePath对应目录下的 views 目录， 对于模块中控制器对应的视图文件应放在 views/ControllerID 目录下， 其中ControllerID对应 [控制器 ID](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes). For example, if 例如，假定控制器类为PostController， 目录对应模块yii\base\Module::basePath目录下的 views/post 目录。

模块可指定 [布局](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts)，它用在模块的控制器视图渲染。 布局文件默认放在 views/layouts 目录下， 可配置[yii\base\Module::$layout](http://www.yiichina.com/doc/api/2.0/yii-base-module#$layout-detail)属性指定布局名， 如果没有配置 layout 属性名，默认会使用应用的布局。

6、模块中的控制台命令

您的模块也可以声明命令，这将通过 [控制台](http://www.yiichina.com/doc/guide/2.0/tutorial-console) 模式可用。

当 Yii 在控制台模式下执行并将其指向命令的命名空间时。想要在命令行中查看你的命令， 你需要更改 [yii\base\Module::$controllerNamespace](http://www.yiichina.com/doc/api/2.0/yii-base-module#$controllerNamespace-detail) 属性。

一种实现方法是在模块的 init() 方法中测试Yii应用程序的实例类型：

**public** **function** **init**()

{

**parent**::init();

**if** (Yii::$app **instanceof** \yii\console\Application) {

$this->controllerNamespace = 'app\modules\forum\commands';

}

}

然后您的命令将从命令行使用以下路由：

yii <**module\_id**>/<**command**>/<**sub\_command**>

7、路由

和访问应用的控制器类似，[路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes) 也用在模块中控制器的寻址， 模块中控制器的路由必须以模块ID开始，接下来为控制器ID和操作ID。 例如，假定应用使用一个名为 forum 模块， 路由forum/post/index 代表模块中 post 控制器的 index 操作， 如果路由只包含模块ID，默认为 default 的 [yii\base\Module::$defaultRoute](http://www.yiichina.com/doc/api/2.0/yii-base-module#$defaultRoute-detail) （可在系统配置中配置）属性来决定使用哪个控制器/操作， 也就是说路由 forum 可能代表 forum 模块的 default 控制器。

8、访问模块

在模块中，可能经常需要获取[模块类](http://www.yiichina.com/doc/guide/2.0/structure-modules#module-classes)的实例来访问模块ID，模块参数，模块组件等， 可以使用如下语句来获取：

$module = MyModuleClass::getInstance();

其中 MyModuleClass 对应你想要的模块类， getInstance() 方法返回当前请求的模块类实例， 如果模块没有被请求，该方法会返回空，注意不需要手动创建一个模块类， 因为手动创建的和Yii处理请求时自动创建的不同。

**信息：**当开发模块时，你不能假定模块使用固定的ID， 因为在应用或其他没模块中，模块可能会对应到任意的ID， 为了获取模块ID，应使用上述代码获取模块实例， 然后通过$module->id获取模块ID。

Exp :1☺获取模块id

$module = MyModuleClass::getInstance();

$id = $module->id;

也可以使用如下方式访问模块实例:

Exp :2☺// 获取ID为 "forum" 的模块

$module = \Yii::$app->getModule('forum');

Exp :3☺// 获取处理当前请求控制器所属的模块

$module = \Yii::$app->controller->module;

第一种方式仅在你知道模块ID的情况下有效， 第二种方式在你知道处理请求的控制器下使用。

一旦获取到模块实例，可访问注册到模块的参数和组件，例如：

$maxPostCount = $module->params['maxPostCount'];

**信息：**[getModule()](http://www.yiichina.com/doc/api/2.0/yii-base-module#getModule()-detail) 方法只返回子模块的直属的 父模块。 [yii\base\Application::$loadedModules](http://www.yiichina.com/doc/api/2.0/yii-base-application#$loadedModules-detail) 保存了已加载的所有模块的属性, 包括两者的子模块和 嵌套模块，并用他们的类名进行索引。

9、引导启动模块

有些模块在每个请求下都有运行， yii\debug\Module 模块就是这种， 为此将这种模块加入到应用主体的 [bootstrap](http://www.yiichina.com/doc/api/2.0/yii-base-application#bootstrap()-detail) 属性中。

例如，如下示例的应用主体配置会确保debug模块每次都被加载：

[

'bootstrap' => [

'debug',

],

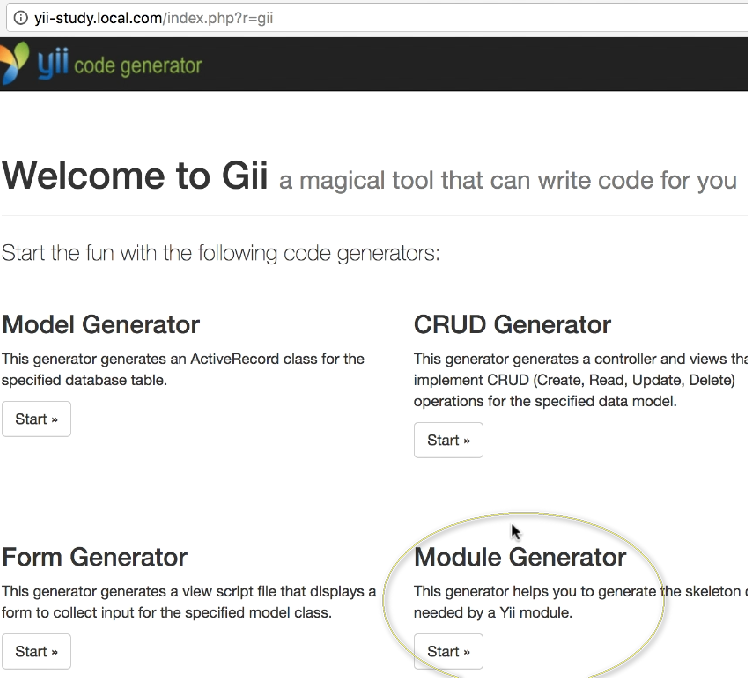
'modules' => [

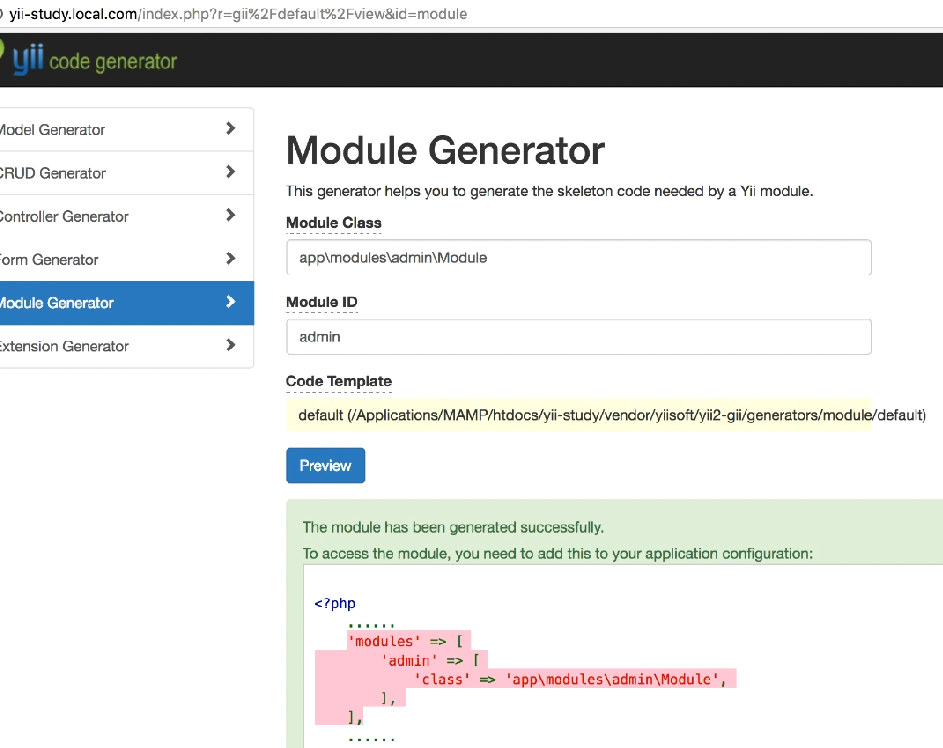
'debug' => 'yii\debug\Module',

],

]

10、Yii2中用Gii生成Module





11、模块与布局

在模块的配置文件中可以设置布局文件，如

$confing = [

‘admin’ => [

‘class’ => ‘app\modules\admin\Module’,

‘layout’ => ‘main’

]

]

12、调用其它模块的方法

在模块中调用其它模块的action();

Exp1\_1:

<?php

Namepace app\modules\admin\controllers;

Use yii\web\Controller;

Class DefaultController extends Controller{

Public function actionIndex(){

Return $this->render(view:’index’,[

‘a’ => ‘hello modeule’

]);

}

Public function actionHello(){

Return ‘我是Default控制器中的’.\_\_FUNCTION\_\_;

}

}

?>

Exp1\_2:

<?php

Namepace app\controllers;

Use yii\web\Controller;

Class Course122Controller extends Controller{

Public function actionModule(){

$admin = Yii::$app -> getModule(id:’admin’);

Echo $admin -> runAction(roule:’default/hello’);

}

}

?>

**十、路径别名**

大Yii2中用别名来表示路径和URL，以避免在代码中硬编码一些绝对路径和URL.

1、预定义别名（用来简化常用路径和 URL）：

* @yii，BaseYii.php 文件所在的目录（也被称为框架安装目录）。
* @app，当前运行的应用 yii\base\Application::basePath。
* @runtime，当前运行的应用的 yii\base\Application::runtimePath。默认为 @app/runtime。
* @webroot，当前运行的Web应用程序的Web根目录。 它是根据包含 [入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) 的目录确定的。
* @web，当前运行的Web应用程序的 base URL。它的值与 yii\web\Request::baseUrl 相同。
* @vendor，yii\base\Application::vendorPath。
* @bower，包含 [bower 包](http://bower.io/) 的根目录。默认为 @vendor/bower。
* @npm，包含 [npm 包](https://www.npmjs.org/) 的根目录。默认为 @vendor/npm。

@yii 别名是在[入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)里包含 Yii.php 文件时定义的， 其他的别名都是在[配置应用](http://www.yiichina.com/doc/guide/2.0/concept-configurations)的时候， 于应用的构造方法内定义的。

**注意：**通过composer安装的扩展会以其在composer.json文件中声明的根命名空间为名自动添加别名。例如：

如果你安装有 yiisoft/yii2-jui 扩展，会自动得到 @yii/jui 别名， 它定义于引导启动阶段： Yii::setAlias('@yii/jui', 'VendorPath/yiisoft/yii2-jui');

2、解析别名

调用Yii::getAlias()解析 根别名 和 衍生别名。

Exp 1：

**echo** Yii::getAlias('@foo'); // 输出：/path/to/foo

**echo** Yii::getAlias('@bar'); // 输出：http://www.example.com

**echo** Yii::getAlias('@foo/bar/file.php'); // 输出：/path/to/foo/bar/file.php

**注意：1、**由衍生别名所解析出的文件路径和 URL 是通过替换掉衍生别名中的根别名部分得到的。

2、Yii::getAlias() 并不检查结果路径（/URL） 所指向的资源是否真实存在。

3、在编程过程中，Yii能正确识别别名，如：

$cache = new FileCache([

'cachePath' => '@runtime/cache',

]);

4、通过Yii::$aliases查看已经定义的别名。

2、查看系统中定义的别名：

print\_r(Yii::$aliases);

设置别名：Yii::setAlias($alias, $path)

**十一、过滤器**

过滤器是 控制器动作 执行之前（预过滤）或之后（后过滤）执行的对象。

可在控制器类中覆盖yii\base\Controller::behaivors()方法来申明过滤器。如：

**public** **function** **behaviors**()

{

**return** [

[

'class' => 'yii\filters\HttpCache',

'only' => ['index', 'view'],

'lastModified' => **function** ($action, $params) {

$q = **new** \yii\db\Query();

**return** $q->from('user')->max('updated\_at');

},

],

];

}

控制器类的过滤器默认应用到该类的 所有 动作， 你可以配置[only](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$only-detail)属性明确指定控制器应用到哪些动作。 在上述例子中，HttpCache 过滤器只应用到index和view动作。 也可以配置[except](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$except-detail)属性 使一些动作不执行过滤器。

除了控制器外，可在 [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules)或[应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications) 中申明过滤器。 申明之后，过滤器会应用到所属该模块或应用主体的 所有 控制器动作， 除非像上述一样配置过滤器的 [only](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$only-detail) 和 [except](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$except-detail) 属性。

**注意：**在模块或应用主体中申明过滤器，在[only](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$only-detail) 和 [except](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#$except-detail) 属性中使用[路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes) 代替动作ID， 因为在模块或应用主体中只用动作ID并不能唯一指定到具体动作。

* 预过滤
  + 按顺序执行应用主体中behaviors()列出的过滤器。
  + 按顺序执行模块中behaviors()列出的过滤器。
  + 按顺序执行控制器中behaviors()列出的过滤器。
  + 如果任意过滤器终止动作执行， 后面的过滤器（包括预过滤和后过滤）不再执行。
* 成功通过预过滤后执行动作。
* 后过滤
  + 倒序执行控制器中behaviors()列出的过滤器。
  + 倒序执行模块中behaviors()列出的过滤器。
  + 倒序执行应用主体中behaviors()列出的过滤器。

1. 创建过滤器

继承 [yii\base\ActionFilter](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter) 类并覆盖 [beforeAction()](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#beforeAction()-detail) 或 [afterAction()](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#afterAction()-detail) 方法来创建动作的过滤器，前者在动作执行之前执行，后者在动作执行之后执行。 [beforeAction()](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#beforeAction()-detail) 返回值决定动作是否应该执行， 如果为 false，之后的过滤器和动作不会继续执行。

下面的例子申明一个记录动作执行时间日志的过滤器。

**namespace** **app**\**components**;

**use** **Yii**;

**use** **yii**\**base**\**ActionFilter**;

**class** **ActionTimeFilter** **extends** **ActionFilter**

{

**private** $\_startTime;

**public** **function** **beforeAction**($action)

{

$this->\_startTime = microtime(**true**);

**return** **parent**::beforeAction($action);

}

**public** **function** **afterAction**($action, $result)

{

$time = microtime(**true**) - $this->\_startTime;

Yii::trace("Action '{$action->uniqueId}' spent $time second.");

**return** **parent**::afterAction($action, $result);

}

}

2、核心过滤器

2.1、访问控制过滤器AccessControl

AccessControl 提供基于 [rules](http://www.yiichina.com/doc/api/2.0/yii-filters-accesscontrol#$rules-detail) 规则的访问控制。 特别是在动作执行之前，访问控制会检测所有规则 并找到第一个符合上下文的变量（比如用户IP地址、登录状态等等）的规则， 来决定允许还是拒绝请求动作的执行， 如果没有规则符合，访问就会被拒绝。

如下示例表示表示允许已认证用户访问 create 和 update 动作， 拒绝其他用户访问这两个动作。

**use** **yii**\**filters**\**AccessControl**;

**public** **function** **behaviors**()

{

**return** [

'access' => [

'class' => AccessControl::className(),

'only' => ['create', 'update'],

'rules' => [

// 允许认证用户

[

'allow' => **true**,

'roles' => ['@'],

],

// 默认禁止其他用户

],

],

];

}

更多关于访问控制的详情请参阅 [授权](http://www.yiichina.com/doc/guide/2.0/security-authorization) 一节。

2.2、认证方法过滤器

认证方法过滤器通过[HTTP Basic Auth](http://en.wikipedia.org/wiki/Basic_access_authentication) 或[OAuth 2](http://oauth.net/2/) 来认证一个用户，认证方法过滤器类在 yii\filters\auth 命名空间下。

如下示例表示可使用[yii\filters\auth\HttpBasicAuth](http://www.yiichina.com/doc/api/2.0/yii-filters-auth-httpbasicauth)来认证一个用户， 它使用基于HTTP基础认证方法的令牌。 注意为了可运行，[user identity class](http://www.yiichina.com/doc/api/2.0/yii-web-user#$identityClass-detail) 类必须 实现 [findIdentityByAccessToken()](http://www.yiichina.com/doc/api/2.0/yii-web-identityinterface#findIdentityByAccessToken()-detail)方法。

**use** **yii**\**filters**\**auth**\**HttpBasicAuth**;

**public** **function** **behaviors**()

{

**return** [

'basicAuth' => [

'class' => HttpBasicAuth::className(),

],

];

}

认证方法过滤器通常在实现RESTful API中使用， 更多关于访问控制的详情请参阅 RESTful [认证](http://www.yiichina.com/doc/guide/2.0/rest-authentication) 一节。

2.3、响应内容格式处理和语言处理

ContentNegotiator支持响应内容格式处理和语言处理。 通过检查 GET 参数和 Accept HTTP头部来决定响应内容格式和语言。

如下示例，配置ContentNegotiator支持JSON和XML 响应格式和英语（美国）和德语。

**use** **yii**\**filters**\**ContentNegotiator**;

**use** **yii**\**web**\**Response**;

**public** **function** **behaviors**()

{

**return** [

[

'class' => ContentNegotiator::className(),

'formats' => [

'application/json' => Response::FORMAT\_JSON,

'application/xml' => Response::FORMAT\_XML,

],

'languages' => [

'en-US',

'de',

],

],

];

}

在[应用主体生命周期](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-lifecycle)过程中检测响应格式和语言简单很多， 因此ContentNegotiator设计可被 [引导启动组件](http://www.yiichina.com/doc/guide/2.0/structure-applications#bootstrap)调用的过滤器。 如下例所示可以将它配置在 [应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations)。

**use** **yii**\**filters**\**ContentNegotiator**;

**use** **yii**\**web**\**Response**;

[

'bootstrap' => [

[

'class' => ContentNegotiator::className(),

'formats' => [

'application/json' => Response::FORMAT\_JSON,

'application/xml' => Response::FORMAT\_XML,

],

'languages' => [

'en-US',

'de',

],

],

],

];

**信息：**如果请求中没有检测到内容格式和语言， 使用 formats 和 languages 第一个配置项。

2.4、HttpCache

HttpCache 利用 Last-Modified 和 Etag HTTP头实现客户端缓存。 例如：

**use** **yii**\**filters**\**HttpCache**;

**public** **function** **behaviors**()

{

**return** [

[

'class' => HttpCache::className(),

'only' => ['index'],

'lastModified' => **function** ($action, $params) {

$q = **new** \yii\db\Query();

**return** $q->from('user')->max('updated\_at');

},

],

];

}

更多关于使用 HttpCache 详情请参阅 [HTTP 缓存](http://www.yiichina.com/doc/guide/2.0/caching-http) 一节。

2.5、PageCache

PageCache 实现服务器端整个页面的缓存。如下示例所示，PageCache应用在 index 动作， 缓存整个页面 60 秒或 post 表的记录数发生变化。 它也会根据不同应用语言保存不同的页面版本。

**use** **yii**\**filters**\**PageCache**;

**use** **yii**\**caching**\**DbDependency**;

**public** **function** **behaviors**()

{

**return** [

'pageCache' => [

'class' => PageCache::className(),

'only' => ['index'],

'duration' => 60,

'dependency' => [

'class' => DbDependency::className(),

'sql' => 'SELECT COUNT(\*) FROM post',

],

'variations' => [

\Yii::$app->language,

]

],

];

}

更多关于使用PageCache详情请参阅 [页面缓存](http://www.yiichina.com/doc/guide/2.0/caching-page) 一节。

2.6、漏桶算法RateLimiter

RateLimiter 根据 [漏桶算法](http://en.wikipedia.org/wiki/Leaky_bucket) 来实现速率限制。 主要用在实现RESTful APIs，更多关于该过滤器详情请参阅 [Rate Limiting](http://www.yiichina.com/doc/guide/2.0/rest-rate-limiting) 一节。

2.7、VerbFilter

VerbFilter检查请求动作的HTTP请求方式是否允许执行， 如果不允许，会抛出HTTP 405异常。 如下示例，VerbFilter指定CRUD动作所允许的请求方式。

**use** **yii**\**filters**\**VerbFilter**;

**public** **function** **behaviors**()

{

**return** [

'verbs' => [

'class' => VerbFilter::className(),

'actions' => [

'index' => ['get'],

'view' => ['get'],

'create' => ['get', 'post'],

'update' => ['get', 'put', 'post'],

'delete' => ['post', 'delete'],

],

],

];

}

2.8、跨域资源共享Cors

跨域资源共享 [CORS](https://developer.mozilla.org/fr/docs/HTTP/Access_control_CORS) 机制允许一个网页的许多资源（例如字体、JavaScript等） 这些资源可以通过其他域名访问获取。 特别是 JavaScript 的 AJAX 调用可使用 XMLHttpRequest 机制， 由于同源安全策略该跨域请求会被网页浏览器禁止. CORS定义浏览器和服务器交互时哪些跨域请求允许和禁止。

[Cors filter](http://www.yiichina.com/doc/api/2.0/yii-filters-cors) 应在 授权 / 认证 过滤器之前定义， 以保证CORS头部被发送。

**use** **yii**\**filters**\**Cors**;

**use** **yii**\**helpers**\**ArrayHelper**;

**public** **function** **behaviors**()

{

**return** ArrayHelper::merge([

[

'class' => Cors::className(),

],

], **parent**::behaviors());

}

如果要将CORS过滤器添加到你的 API 中的 [yii\rest\ActiveController](http://www.yiichina.com/doc/api/2.0/yii-rest-activecontroller) 类， 还要检查 [REST Controllers](http://www.yiichina.com/doc/guide/2.0/rest-controllers.md%EF%BC%83cors) 中的部分。

CROS过滤器可以通过 [$cors](http://www.yiichina.com/doc/api/2.0/yii-filters-cors#$cors-detail) 属性进行调整。

* cors['Origin']：定义允许来源的数组，可为 ['\*']（任何用户）或 ['http://www.myserver.net', 'http://www.myotherserver.com']。 默认为 ['\*']。
* cors['Access-Control-Request-Method']：允许动作数组如 ['GET', 'OPTIONS', 'HEAD']。默认为 ['GET', 'POST', 'PUT', 'PATCH', 'DELETE', 'HEAD', 'OPTIONS']。
* cors['Access-Control-Request-Headers']：允许请求头部数组，可为 ['\*'] 所有类型头部 或 ['X-Request-With'] 指定类型头部。默认为 ['\*']。
* cors['Access-Control-Allow-Credentials']：定义当前请求是否使用证书，可为 true, false 或 null (不设置). 默认为 null。
* cors['Access-Control-Max-Age']: 定义请求的有效时间，默认为 86400。

例如，允许来源为 http://www.myserver.net 和方式为 GET，HEAD 和 OPTIONS 的 CORS 如下：

**use** **yii**\**filters**\**Cors**;

**use** **yii**\**helpers**\**ArrayHelper**;

**public** **function** **behaviors**()

{

**return** ArrayHelper::merge([

[

'class' => Cors::className(),

'cors' => [

'Origin' => ['http://www.myserver.net'],

'Access-Control-Request-Method' => ['GET', 'HEAD', 'OPTIONS'],

],

],

], **parent**::behaviors());

}

可以覆盖默认参数为每个动作调整CORS 头部。例如，为 login 动作 增加 Access-Control-Allow-Credentials参数如下所示：

**use** **yii**\**filters**\**Cors**;

**use** **yii**\**helpers**\**ArrayHelper**;

**public** **function** **behaviors**()

{

**return** ArrayHelper::merge([

[

'class' => Cors::className(),

'cors' => [

'Origin' => ['http://www.myserver.net'],

'Access-Control-Request-Method' => ['GET', 'HEAD', 'OPTIONS'],

],

'actions' => [

'login' => [

'Access-Control-Allow-Credentials' => **true**,

]

]

],

], **parent**::behaviors());

}

十二、请求处理

1、启动引导

在应用主体的构造函数中，会执行以下引导工作：

1. 调用 [preInit()](http://www.yiichina.com/doc/api/2.0/yii-base-application#preInit()-detail)（预初始化）方法，配置一些高优先级的应用属性， 比如 yii\base\Application::basePath 属性。
2. 注册yii\base\Application::errorHandler。
3. 通过给定的应用配置初始化应用的各属性。
4. 通过调用 [init()](http://www.yiichina.com/doc/api/2.0/yii-base-application#init()-detail)（初始化）方法，它会顺次调用 [bootstrap()](http://www.yiichina.com/doc/api/2.0/yii-base-application#bootstrap()-detail) 从而运行引导组件。
   * 加载扩展清单文件(extension manifest file) vendor/yiisoft/extensions.php。
   * 创建并运行各个扩展声明的 [引导组件（bootstrap components）](http://www.yiichina.com/doc/guide/2.0/structure-extensions#bootstrapping-classes)。
   * 创建并运行各个 [应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components) 以及在应用的 [Bootstrap 属性](http://www.yiichina.com/doc/guide/2.0/structure-applications#bootstrap)中声明的各个 [模块（modules）组件](http://www.yiichina.com/doc/guide/2.0/structure-modules)（如果有）。

因为引导工作必须在处理**每一次**请求之前都进行一遍，因此让该过程尽可能轻量化就异常重要， 请尽可能地优化这一步骤。

1. 路由引导与创建URL

2.1、URL Formats

Yii2 的URL管理器支持两种形式的url，及dufault URL 和 pretty URL

Default URL形式如：/index.php?r=post/view&id=100

Pretty URL 形式如：/post/view?id=100

2.2、缺省路由

如果传入请求并没有提供一个具体的路由，此时就会启用由yii\web\Application::$defaultRoute属性所指定的缺省路由。可以在配置中调整该属性的值：

**return** [

// ...

'defaultRoute' => 'site/index',

];

2.3、catchAll全拦截路由

有时候，你会想要将你的 Web 应用临时调整到维护模式，所有的请求下都会显示相同的信息页。当然，要实现这一点有很多种方法。这里面最简单快捷的方法就是在应用配置中设置下 [yii\web\Application::$catchAll](http://www.yiichina.com/doc/api/2.0/yii-web-application#$catchAll-detail) 属性：

**return** [

// ...

'catchAll' => ['site/offline'],

];.

2.4、使用美化url（Pretty URL）

2.4.1、使用pretty URLs，需要在应用主体配置中对$components进行如下配置：

[

'components' => [

'urlManager' => [ //配置urlManager

'enablePrettyUrl' => **true**,

'showScriptName' => **false**,

'enableStrictParsing' => **false**,

'rules' => [

//配置路由规则

'admin'=>'/adminindex/default/index',

'tradeinfo'=>'/trades/list',

'/game' => '/games/list',

//以下为动态路由

'/game/<spelling:\w+>-<gameid:\d+>' => '/trades/list-trade',

'/game/<spelling:\w+>-<gameid:\d+>-<parentgoodsid:\d+>-<goodsid:\d+>' => '/trades/list-trade',

'store\_<id:\d+>' => '/shopinfo',

'/news/<gameid:\d+>-<page:\d+>' => '/download/news',

'news\_<id:\d+>.html' => '/download/news/view',

'/promote/route/<pcid:\d+>' => '/promote/route',

],

],

],

]

注：使用动态路由需要为其创建相匹配的路由规则。

2.4.2、创建URL规则：

**namespace** **app**\**components**;

**use** **yii**\**web**\**UrlRuleInterface**;

**use** **yii**\**base**\**Object**;

**class** **CarUrlRule** **extends** **Object** **implements** **UrlRuleInterface**

{

**public** **function** **createUrl**($manager, $route, $params)

{

**if** ($route === 'car/index') {

**if** (**isset**($params['manufacturer'], $params['model'])) {

**return** $params['manufacturer'] . '/' . $params['model'];

} **elseif** (**isset**($params['manufacturer'])) {

**return** $params['manufacturer'];

}

}

**return** **false**; // this rule does not apply

}

**public** **function** **parseRequest**($manager, $request)

{

$pathInfo = $request->getPathInfo();

**if** (preg\_match('%^(\w+)(/(\w+))?$%', $pathInfo, $matches)) {

// check $matches[1] and $matches[3] to see

// if they match a manufacturer and a model in the database

// If so, set $params['manufacturer'] and/or $params['model']

// and return ['car/index', $params]

}

**return** **false**; // this rule does not apply

}

}

2.4.3、在应用配置中对 [yii\web\UrlManager::$rules](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#$rules-detail) 进行配置:

[

// ...other rules...

[

'class' => 'app\components\CarUrlRule',

// ...configure other properties...

],

]

1. 请求（Request）

3.1、获取请求参数

要获取请求参数，你可以调用 request 组件的 [get()](http://www.yiichina.com/doc/api/2.0/yii-web-request#get()-detail) 方法和 [post()](http://www.yiichina.com/doc/api/2.0/yii-web-request#post()-detail) 方法。 他们分别返回 $\_GET 和 $\_POST 的值。例如，

$request = Yii::$app->request;

$get = $request->get();

// 等价于: $get = $\_GET;

$id = $request->get('id');

// 等价于: $id = isset($\_GET['id']) ? $\_GET['id'] : null;

$id = $request->get('id', 1);

// 等价于: $id = isset($\_GET['id']) ? $\_GET['id'] : 1;

$post = $request->post();

// 等价于: $post = $\_POST;

$name = $request->post('name');

// 等价于: $name = isset($\_POST['name']) ? $\_POST['name'] : null;

$name = $request->post('name', '');

// 等价于: $name = isset($\_POST['name']) ? $\_POST['name'] : '';

**信息：**建议你像上面那样通过 request 组件来获取请求参数，而不是 直接访问 $\_GET 和 $\_POST。 这使你更容易编写测试用例，因为你可以伪造数据来创建一个模拟请求组件。

当实现 [RESTful APIs](http://www.yiichina.com/doc/guide/2.0/rest-quick-start) 接口的时候，你经常需要获取通过PUT， PATCH或者其他的 [request methods](http://www.yiichina.com/doc/guide/2.0/runtime-requests#request-methods) 请求方法提交上来的参数。你可以通过调用 [yii\web\Request::getBodyParam()](http://www.yiichina.com/doc/api/2.0/yii-web-request#getBodyParam()-detail) 方法来获取这些参数。例如，

$request = Yii::$app->request;

// 返回所有参数

$params = $request->bodyParams;

// 返回参数 "id"

$param = $request->getBodyParam('id');

**信息：**不同于 GET 参数，POST，PUT，PATCH 等等这些提交上来的参数是在请求体中被发送的。 当你通过上面介绍的方法访问这些参数的时候，request 组件会解析这些参数。 你可以通过配置 [yii\web\Request::$parsers](http://www.yiichina.com/doc/api/2.0/yii-web-request#$parsers-detail) 属性来自定义怎样解析这些参数。

3.2请求方法

通过 Yii::$app->request->method 表达式来获取当前请求使用的HTTP方法。 这里还提供了一整套布尔属性用于检测当前请求是某种类型。 例如，

$request = Yii::$app->request;

**if** ($request->isAjax) { /\* 该请求是一个 AJAX 请求 \*/ }

**if** ($request->isGet) { /\* 请求方法是 GET \*/ }

**if** ($request->isPost) { /\* 请求方法是 POST \*/ }

**if** ($request->isPut) { /\* 请求方法是 PUT \*/ }

3.3、获取请求URLs

request 组件提供了许多方式来检测当前请求的URL。

假设被请求的URL是 http://example.com/admin/index.php/product?id=100， 你可以像下面描述的那样获取URL的各个部分：

* yii\web\Request::url：返回 /admin/index.php/product?id=100, 此URL不包括host info部分。
* yii\web\Request::absoluteUrl：返回 http://example.com/admin/index.php/product?id=100, 包含host infode的整个URL。
* yii\web\Request::hostInfo：返回 http://example.com, 只有host info部分。
* yii\web\Request::pathInfo：返回 /product， 这个是入口脚本之后，问号之前（查询字符串）的部分。
* yii\web\Request::queryString：返回 id=100,问号之后的部分。
* yii\web\Request::baseUrl：返回 /admin, host info之后， 入口脚本之前的部分。
* yii\web\Request::scriptUrl：返回 /admin/index.php, 没有path info和查询字符串部分。
* yii\web\Request::serverName：返回 example.com, URL中的host name。
* yii\web\Request::serverPort：返回 80, 这是web服务中使用的端口。

3.4、获取请求头

你可以通过 yii\web\Request::headers 属性返回的 [header collection](http://www.yiichina.com/doc/api/2.0/yii-web-headercollection) 获取HTTP头信息。 例如，

// $headers 是一个 yii\web\HeaderCollection 对象

$headers = Yii::$app->request->headers;

// 返回 Accept header 值

$accept = $headers->get('Accept');

**if** ($headers->has('User-Agent')) { /\* 这是一个 User-Agent 头 \*/ }

请求组件也提供了支持快速访问常用头的方法，包括：

* yii\web\Request::userAgent：返回 User-Agent 头。
* yii\web\Request::contentType：返回 Content-Type 头的值， Content-Type 是请求体中MIME类型数据。
* yii\web\Request::acceptableContentTypes：返回用户可接受的内容MIME类型。 返回的类型是按照他们的质量得分来排序的。得分最高的类型将被最先返回。
* yii\web\Request::acceptableLanguages：返回用户可接受的语言。 返回的语言是按照他们的偏好层次来排序的。第一个参数代表最优先的语言。

假如你的应用支持多语言，并且你想在终端用户最喜欢的语言中显示页面， 那么你可以使用语言协商方法 [yii\web\Request::getPreferredLanguage()](http://www.yiichina.com/doc/api/2.0/yii-web-request#getPreferredLanguage()-detail)。 这个方法通过 yii\web\Request::acceptableLanguages 在你的应用中所支持的语言列表里进行比较筛选，返回最适合的语言。

**提示：**你也可以使用 [ContentNegotiator](http://www.yiichina.com/doc/api/2.0/yii-filters-contentnegotiator) 过滤器进行动态确定哪些内容类型和语言应该在响应中使用。 这个过滤器实现了上面介绍的内容协商的属性和方法。

3.5、获取客户端信息

你可以通过 yii\web\Request::userHost 和 yii\web\Request::userIP 分别获取host name和客户机的IP地址， 例如，

$userHost = Yii::$app->request->userHost;

$userIP = Yii::$app->request->userIP;

4、响应（Responses）

当应用完成处理一个请求后，会生成一个responses响应对象并发送给终端用户。

响应对象包含以下信息：

HTTP状态码、HTTP头、主体内容。

4.1、状态码

构建响应时，使用一个有效的HTTP状态码为yii\web\Response::statusCode属性的值。如下标识处理已被处理成功：

Yii::$app->response->statusCode = 200;

指定请求失败，可抛出对应的HTTP异常，如下所示：

**throw** **new** \yii\web\NotFoundHttpException;

当[错误处理器](http://www.yiichina.com/doc/guide/2.0/runtime-handling-errors) 捕获到一个异常，会从异常中提取状态码并赋值到响应， 对于上述的 [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception) 对应HTTP 404状态码， 以下为Yii预定义的HTTP异常：

* [yii\web\BadRequestHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-badrequesthttpexception)：状态码 400。
* [yii\web\ConflictHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-conflicthttpexception)：状态码 409。
* [yii\web\ForbiddenHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-forbiddenhttpexception)：状态码 403。
* [yii\web\GoneHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-gonehttpexception)：状态码 410。
* [yii\web\MethodNotAllowedHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-methodnotallowedhttpexception)：状态码 405。
* [yii\web\NotAcceptableHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notacceptablehttpexception)：状态码 406。
* [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception)：状态码 404。
* [yii\web\ServerErrorHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-servererrorhttpexception)：状态码 500。
* [yii\web\TooManyRequestsHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-toomanyrequestshttpexception)：状态码 429。
* [yii\web\UnauthorizedHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-unauthorizedhttpexception)：状态码 401。
* [yii\web\UnsupportedMediaTypeHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-unsupportedmediatypehttpexception)：状态码 415。

如果想抛出的异常不在如上列表中，可创建一个[yii\web\HttpException](http://www.yiichina.com/doc/api/2.0/yii-web-httpexception)异常， 带上状态码抛出，如下：

**throw** **new** \yii\web\HttpException(402);

4.2、HTTP头部

可在 response 组件中操控yii\web\Response::headers来发送HTTP头部信息， 例如：

$headers = Yii::$app->response->headers;

// 增加一个 Pragma 头，已存在的Pragma 头不会被覆盖。

$headers->add('Pragma', 'no-cache');

// 设置一个Pragma 头. 任何已存在的Pragma 头都会被丢弃

$headers->set('Pragma', 'no-cache');

// 删除Pragma 头并返回删除的Pragma 头的值到数组

$values = $headers->remove('Pragma');

**信息：**头名称是大小写敏感的，在[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail)方法 调用前新注册的头信息并不会发送给用户。

4.3、响应主体

响应主体存放你想要显示给终端用户的内容。

如果主体字符串已经格式化，可赋值到响应的[yii\web\Response::$content](http://www.yiichina.com/doc/api/2.0/yii-web-response#$content-detail)属性， 例如：

Yii::$app->response->content = 'hello world!';

如果在发送给终端用户之前需要格式化，应设置 [format](http://www.yiichina.com/doc/api/2.0/yii-web-response#$format-detail) 和 [data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail) 属性，[format](http://www.yiichina.com/doc/api/2.0/yii-web-response#$format-detail) 属性指定[data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail)中数据格式化后的样式，例如：

$response = Yii::$app->response;

$response->format = \yii\web\Response::FORMAT\_JSON;

$response->data = ['message' => 'hello world'];

Yii支持以下格式，他们各自实现了[formatter](http://www.yiichina.com/doc/api/2.0/yii-web-responseformatterinterface) 类， 也可自定义这些格式器或通过配置[yii\web\Response::$formatters](http://www.yiichina.com/doc/api/2.0/yii-web-response#$formatters-detail) 属性来增加格式器。

* [HTML](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_HTML-detail): 通过 [yii\web\HtmlResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-htmlresponseformatter) 来实现.
* [XML](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_XML-detail): 通过 [yii\web\XmlResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-xmlresponseformatter)来实现.
* [JSON](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_JSON-detail): 通过 [yii\web\JsonResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-jsonresponseformatter)来实现.
* [JSONP](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_JSONP-detail): 通过 [yii\web\JsonResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-jsonresponseformatter)来实现.
* [RAW](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_RAW-detail): use this format if you want to send the response directly without applying any formatting.

上述响应主体可明确地设置，但是在大多数情况下是通过 [操作](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 方法的返回值隐式地设置的，常用场景如下所示：

**public** **function** **actionIndex**()

{

**return** $this->render('index');

}

上述的 index 操作返回 index 视图渲染结果， 返回值会被 response 组件格式化后发送给终端用户。

因为响应格式默认为[HTML](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_HTML-detail), 只需要在操作方法中返回一个字符串， 如果想使用其他响应格式，应在返回数据前先设置格式， 例如：

**public** **function** **actionInfo**()

{

//实现内容主体被格式化的代码如下：

\Yii::$app->response->format = \yii\web\Response::FORMAT\_JSON;

**return** [

'message' => 'hello world',

'code' => 100,

];

}

如上所述，触雷使用默认的 response 应用组件，也可创建自己的响应对象并发送给终端用户， 可在操作方法中返回该响应对象，如下所示：

**public** **function** **actionInfo**()

{

**return** \Yii::createObject([

//自定义格式化格式

'class' => 'yii\web\Response',

'format' => \yii\web\Response::FORMAT\_JSON,

'data' => [

'message' => 'hello world',

'code' => 100,

],

]);

}

**注意：**如果创建你自己的响应对象，将不能在应用配置中设置 response 组件，尽管如此， 可使用 [依赖注入](http://www.yiichina.com/doc/guide/2.0/concept-di-container) 应用通用配置到你新的响应对象。

4.4、浏览器跳转

浏览器跳转依赖于发送一个Location HTTP 头，因为该功能通常被使用， Yii提供对它提供了特别的支持。

可调用[yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect()-detail) 方法将用户浏览器跳转到一个URL地址，该方法设置合适的 带指定URL的 Location 头并返回它自己为响应对象，在操作的方法中， 可调用缩写版[yii\web\Controller::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-controller#redirect()-detail)，例如：

**public** **function** **actionOld**()

{

**return** $this->redirect('http://example.com/new', 301);

}

在如上代码中，操作的方法返回redirect() 方法的结果，如前所述， 操作的方法返回的响应对象会被当总响应发送给终端用户。

除了动作方法外，可直接调用[yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect()-detail) 再调用 [yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail) 方法来确保没有其他内容追加到响应中。

\Yii::$app->response->redirect('http://example.com/new', 301)->send();

**信息：**[yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect()-detail) 方法默认会设置响应状态码为302，该状态码会告诉浏览器请求的资源 临时放在另一个URI地址上，可传递一个301状态码告知浏览器请求 的资源已经 永久 重定向到新的URId地址。

如果当前请求为AJAX 请求， 发送一个 Location 头不会自动使浏览器跳转，为解决这个问题，[yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect()-detail) 方法设置一个值为要跳转的URL的X-Redirect 头， 在客户端可编写JavaScript 代码读取该头部值然后让浏览器跳转对应的URL。

**信息：**Yii 配备了一个yii.js JavaScript 文件提供常用JavaScript功能， 包括基于X-Redirect头的浏览器跳转， 因此，如果你使用该JavaScript 文件(通过[yii\web\YiiAsset](http://www.yiichina.com/doc/api/2.0/yii-web-yiiasset) 资源包注册)， 就不需要编写AJAX跳转的代码。

4.5、发送文件

和浏览器跳转类似，文件发送是另一个依赖指定HTTP头的功能， Yii提供方法集合来支持各种文件发送需求，它们对HTTP头都有内置的支持。

这些方法都将响应对象作为返回值，如果要发送的文件非常大，应考虑使用 [yii\web\Response::sendStreamAsFile()](http://www.yiichina.com/doc/api/2.0/yii-web-response#sendStreamAsFile()-detail) 因为它更节约内存， 以下示例显示在控制器操作中如何发送文件：

**public** **function** **actionDownload**()

{

**return** \Yii::$app->response->sendFile('path/to/file.txt');

}

如果不是在操作方法中调用文件发送方法，在后面还应调用 [yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail) 没有其他内容追加到响应中。

\Yii::$app->response->sendFile('path/to/file.txt')->send();

一些浏览器提供特殊的名为X-Sendfile的文件发送功能， 原理为将请求跳转到服务器上的文件， Web应用可在服务器发送文件前结束，为使用该功能， 可调用[yii\web\Response::xSendFile()](http://www.yiichina.com/doc/api/2.0/yii-web-response#xSendFile()-detail)， 如下简要列出一些常用Web服务器如何启用X-Sendfile 功能：

* Apache: [X-Sendfile](http://tn123.org/mod_xsendfile)
* Lighttpd v1.4: [X-LIGHTTPD-send-file](http://redmine.lighttpd.net/projects/lighttpd/wiki/X-LIGHTTPD-send-file)
* Lighttpd v1.5: [X-Sendfile](http://redmine.lighttpd.net/projects/lighttpd/wiki/X-LIGHTTPD-send-file)
* Nginx: [X-Accel-Redirect](http://wiki.nginx.org/XSendfile)
* Cherokee: [X-Sendfile and X-Accel-Redirect](http://www.cherokee-project.com/doc/other_goodies.html#x-sendfile)

4.6、发送响应

在[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail) 方法调用前响应中的内容不会发送给用户， 该方法默认在[yii\base\Application::run()](http://www.yiichina.com/doc/api/2.0/yii-base-application#run()-detail) 结尾自动调用，尽管如此，可以明确调用该方法强制立即发送响应。

[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail) 方法使用以下步骤来发送响应：

1. 触发 [yii\web\Response::EVENT\_BEFORE\_SEND](http://www.yiichina.com/doc/api/2.0/yii-web-response#EVENT_BEFORE_SEND-detail) 事件.
2. 调用 [yii\web\Response::prepare()](http://www.yiichina.com/doc/api/2.0/yii-web-response#prepare()-detail) 来格式化 [response data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail) 为 [response content](http://www.yiichina.com/doc/api/2.0/yii-web-response#$content-detail).
3. 触发 [yii\web\Response::EVENT\_AFTER\_PREPARE](http://www.yiichina.com/doc/api/2.0/yii-web-response#EVENT_AFTER_PREPARE-detail) 事件.
4. 调用 [yii\web\Response::sendHeaders()](http://www.yiichina.com/doc/api/2.0/yii-web-response#sendHeaders()-detail) 来发送注册的HTTP头
5. 调用 [yii\web\Response::sendContent()](http://www.yiichina.com/doc/api/2.0/yii-web-response#sendContent()-detail) 来发送响应主体内容
6. 触发 [yii\web\Response::EVENT\_AFTER\_SEND](http://www.yiichina.com/doc/api/2.0/yii-web-response#EVENT_AFTER_SEND-detail) 事件.

一旦[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail) 方法被执行后，其他地方调用该方法会被忽略， 这意味着一旦响应发出后，就不能再追加其他内容。

如你所见[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send()-detail) 触发了几个实用的事件， 通过响应这些事件可调整或包装响应。

### 5、Sessions 和 Cookies

1. [Sessions](http://www.yiichina.com/doc/guide/2.0/runtime-sessions-cookies#sessions)
2. [Cookies](http://www.yiichina.com/doc/guide/2.0/runtime-sessions-cookies#cookies)

Sessions 和 cookies 允许数据在多次请求中保持， 在纯 PHP 中，可以分别使用全局变量 $\_SESSION 和 $\_COOKIE 来访问，Yii 将 session 和 cookie 封装成对象并增加一些功能， 可通过面向对象方式访问它们。

#### Sessions

和 [请求](http://www.yiichina.com/doc/guide/2.0/runtime-requests) 和 [响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses)类似， 默认可通过为 [yii\web\Session](http://www.yiichina.com/doc/api/2.0/yii-web-session) 实例的 session [应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components) 来访问 sessions。

##### 开启和关闭 Sessions

可使用以下代码来开启和关闭 session。

$session = Yii::$app->session;

// 检查session是否开启

**if** ($session->isActive) ...

// 开启session

$session->open();

// 关闭session

$session->close();

// 销毁session中所有已注册的数据

$session->destroy();

多次调用[open()](http://www.yiichina.com/doc/api/2.0/yii-web-session#open()-detail) 和[close()](http://www.yiichina.com/doc/api/2.0/yii-web-session#close()-detail) 方法并不会产生错误， 因为方法内部会先检查session是否已经开启。

##### 访问 Session 数据

可使用如下方式访问session中的数据：

$session = Yii::$app->session;

// 获取session中的变量值，以下用法是相同的：

$language = $session->get('language');

$language = $session['language'];

$language = **isset**($\_SESSION['language']) ? $\_SESSION['language'] : **null**;

// 设置一个session变量，以下用法是相同的：

$session->set('language', 'en-US');

$session['language'] = 'en-US';

$\_SESSION['language'] = 'en-US';

// 删除一个session变量，以下用法是相同的：

$session->remove('language');

**unset**($session['language']);

**unset**($\_SESSION['language']);

// 检查session变量是否已存在，以下用法是相同的：

**if** ($session->has('language')) ...

**if** (**isset**($session['language'])) ...

**if** (**isset**($\_SESSION['language'])) ...

// 遍历所有session变量，以下用法是相同的：

**foreach** ($session **as** $name => $value) ...

**foreach** ($\_SESSION **as** $name => $value) ...

**信息：**当使用 session 组件访问 session 数据时候， 如果 session 没有开启会自动开启， 这和通过 $\_SESSION 不同，$\_SESSION 要求先执行 session\_start()。

当 session 数据为数组时，session 组件会限制你直接修改数据中的单元项， 例如：

$session = Yii::$app->session;

// 如下代码不会生效

$session['captcha']['number'] = 5;

$session['captcha']['lifetime'] = 3600;

// 如下代码会生效：

$session['captcha'] = [

'number' => 5,

'lifetime' => 3600,

];

// 如下代码也会生效：

**echo** $session['captcha']['lifetime'];

可使用以下任意一个变通方法来解决这个问题：

$session = Yii::$app->session;

// 直接使用$\_SESSION (确保Yii::$app->session->open() 已经调用)

$\_SESSION['captcha']['number'] = 5;

$\_SESSION['captcha']['lifetime'] = 3600;

// 先获取session数据到一个数组，修改数组的值，然后保存数组到session中

$captcha = $session['captcha'];

$captcha['number'] = 5;

$captcha['lifetime'] = 3600;

$session['captcha'] = $captcha;

// 使用ArrayObject 数组对象代替数组

$session['captcha'] = **new** \ArrayObject;

...

$session['captcha']['number'] = 5;

$session['captcha']['lifetime'] = 3600;

// 使用带通用前缀的键来存储数组

$session['captcha.number'] = 5;

$session['captcha.lifetime'] = 3600;

为更好的性能和可读性，推荐最后一种方案， 也就是不用存储 session 变量为数组， 而是将每个数组项变成有相同键前缀的 session 变量。

##### 自定义Session存储

[yii\web\Session](http://www.yiichina.com/doc/api/2.0/yii-web-session) 类默认存储 session 数据为文件到服务器上， Yii 提供以下 session 类实现不同的 session 存储方式：

* [yii\web\DbSession](http://www.yiichina.com/doc/api/2.0/yii-web-dbsession)：存储 session 数据在数据表中
* [yii\web\CacheSession](http://www.yiichina.com/doc/api/2.0/yii-web-cachesession)：存储 session 数据到缓存中，缓存和配置中的[缓存组件](http://www.yiichina.com/doc/guide/2.0/caching-data#cache-components)相关
* yii\redis\Session：存储 session 数据到以 [redis](http://redis.io/) 作为存储媒介中
* yii\mongodb\Session：存储 session 数据到 [MongoDB](http://www.mongodb.org/)。

所有这些session类支持相同的API方法集，因此， 切换到不同的session存储介质不需要修改项目使用session的代码。

**注意：**如果通过$\_SESSION访问使用自定义存储介质的session， 需要确保session已经用[yii\web\Session::open()](http://www.yiichina.com/doc/api/2.0/yii-web-session#open()-detail) 开启， 这是因为在该方法中注册自定义session存储处理器。

学习如何配置和使用这些组件类请参考它们的API文档，如下为一个示例 显示如何在应用配置中配置[yii\web\DbSession](http://www.yiichina.com/doc/api/2.0/yii-web-dbsession)将数据表作为 session 存储介质。

**return** [

'components' => [

'session' => [

'class' => 'yii\web\DbSession',

// 'db' => 'mydb', // 数据库连接的应用组件ID，默认为'db'.

// 'sessionTable' => 'my\_session', // session 数据表名，默认为'session'.

],

],

];

也需要创建如下数据库表来存储 session 数据：

**CREATE** **TABLE** **session**

(

**id** CHAR(40) **NOT** NULL PRIMARY **KEY**,

**expire** INTEGER,

**data** BLOB

)

其中 'BLOB' 对应你选择的数据库管理系统的 BLOB-type 类型，以下一些常用数据库管理系统的 BLOB 类型：

* MySQL: LONGBLOB
* PostgreSQL: BYTEA
* MSSQL: BLOB

**注意：**根据 php.ini 设置的 session.hash\_function，你需要调整 id 列的长度， 例如，如果 session.hash\_function=sha256， 应使用长度为 64 而不是 40 的 char 类型。

或者，可以通过以下迁移完成：

<?php

**use** **yii**\**db**\**Migration**;

**class** **m170529\_050554\_create\_table\_session** **extends** **Migration**

{

**public** **function** **up**()

{

$this->createTable('{{%session}}', [

'id' => $this->char(64)->notNull(),

'expire' => $this->integer(),

'data' => $this->binary()

]);

$this->addPrimaryKey('pk-id', '{{%session}}', 'id');

}

**public** **function** **down**()

{

$this->dropTable('{{%session}}');

}

}

##### Flash 数据

Flash数据是一种特别的 session 数据，它一旦在某个请求中设置后， 只会在下次请求中有效，然后该数据就会自动被删除。 常用于实现只需显示给终端用户一次的信息， 如用户提交一个表单后显示确认信息。

可通过 session 应用组件设置或访问 session，例如：

$session = Yii::$app->session;

// 请求 #1

// 设置一个名为"postDeleted" flash 信息

$session->setFlash('postDeleted', 'You have successfully deleted your post.');

// 请求 #2

// 显示名为"postDeleted" flash 信息

**echo** $session->getFlash('postDeleted');

// 请求 #3

// $result 为 false，因为flash信息已被自动删除

$result = $session->hasFlash('postDeleted');

和普通 session 数据类似，可将任意数据存储为 flash 数据。

当调用 [yii\web\Session::setFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#setFlash()-detail) 时, 会自动覆盖相同名的已存在的任何数据， 为将数据追加到已存在的相同名 flash 中，可改为调用 [yii\web\Session::addFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#addFlash()-detail)。 例如:

$session = Yii::$app->session;

// 请求 #1

// 在名称为"alerts"的flash信息增加数据

$session->addFlash('alerts', 'You have successfully deleted your post.');

$session->addFlash('alerts', 'You have successfully added a new friend.');

$session->addFlash('alerts', 'You are promoted.');

// 请求 #2

// $alerts 为名为'alerts'的flash信息，为数组格式

$alerts = $session->getFlash('alerts');

**注意：**不要在相同名称的 flash 数据中使用 [yii\web\Session::setFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#setFlash()-detail) 的同时也使用 [yii\web\Session::addFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#addFlash()-detail)， 因为后一个防范会自动将 flash 信息转换为数组以使新的 flash 数据可追加进来，因此， 当你调用 [yii\web\Session::getFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#getFlash()-detail) 时， 会发现有时获取到一个数组，有时获取到一个字符串， 取决于你调用这两个方法的顺序。

**提示：**For displaying Flash messages you can use yii\bootstrap\Alert widget in the following way:

**echo** Alert::widget([

'options' => ['class' => 'alert-info'],

'body' => Yii::$app->session->getFlash('postDeleted'),

]);

#### Cookies

Yii使用 [yii\web\Cookie](http://www.yiichina.com/doc/api/2.0/yii-web-cookie) 对象来代表每个 cookie， [yii\web\Request](http://www.yiichina.com/doc/api/2.0/yii-web-request) 和 [yii\web\Response](http://www.yiichina.com/doc/api/2.0/yii-web-response) 通过名为 'cookies' 的属性维护一个 cookie 集合，前者的 cookie 集合代表请求提交的 cookies， 后者的 cookie 集合表示发送给用户的 cookies。

控制器是直接处理请求和响应的部分。 因此, 应当在控制器中读取和发送 cookie 。 (译者注：意思在控制器中处理 cookie 是安全的。)

##### 读取 Cookies

当前请求的cookie信息可通过如下代码获取：

// 从 "request" 组件中获取 cookie 集合(yii\web\CookieCollection)

$cookies = Yii::$app->request->cookies;

// 获取名为 "language" cookie 的值，如果不存在，返回默认值 "en"

$language = $cookies->getValue('language', 'en');

// 另一种方式获取名为 "language" cookie 的值

**if** (($cookie = $cookies->get('language')) !== **null**) {

$language = $cookie->value;

}

// 可将 $cookies 当作数组使用

**if** (**isset**($cookies['language'])) {

$language = $cookies['language']->value;

}

// 判断是否存在名为"language" 的 cookie

**if** ($cookies->has('language')) ...

**if** (**isset**($cookies['language'])) ...

##### 发送 Cookies

可使用如下代码发送 cookie 到终端用户：

// 从 "response" 组件中获取 cookie 集合(yii\web\CookieCollection)

$cookies = Yii::$app->response->cookies;

// 在要发送的响应中添加一个新的 cookie

$cookies->add(**new** \yii\web\Cookie([

'name' => 'language',

'value' => 'zh-CN',

]));

// 删除一个 cookie

$cookies->remove('language');

// 等同于以下删除代码

**unset**($cookies['language']);

除了上述例子定义的 [name](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$name-detail) 和 [value](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$value-detail) 属性 [yii\web\Cookie](http://www.yiichina.com/doc/api/2.0/yii-web-cookie) 类也定义了其他属性来实现cookie的各种信息，如 [domain](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$domain-detail)，[expire](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$expire-detail) 可配置这些属性到 cookie 中并添加到响应的 cookie 集合中。

**注意：**为安全起见 [yii\web\Cookie::$httpOnly](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$httpOnly-detail) 被设置为 true， 这可减少客户端脚本访问受保护 cookie（如果浏览器支持）的风险， 更多详情可阅读 [httpOnly wiki article](https://www.owasp.org/index.php/HttpOnly)。

##### Cookie验证

在上两节中，当通过 request 和 response 组件读取和发送 cookie 时， 你会喜欢扩展的 cookie 验证的保障安全功能，它能 使 cookie 不被客户端修改。该功能通过给每个cookie签发一个哈希字符串来告知服务端 cookie 是否在客户端被修改， 如果被修改，通过 request 组件的 yii\web\Request::cookies cookie 集合访问不到该 cookie。

**注意：**Cookie验证只保护 cookie 值被修改，如果一个 cookie 验证失败， 仍然可以通过 $\_COOKIE 来访问该 cookie， 因为这是第三方库对未通过 cookie 验证自定义的操作方式。

Cookie 验证默认启用，可以设置 [yii\web\Request::$enableCookieValidation](http://www.yiichina.com/doc/api/2.0/yii-web-request#$enableCookieValidation-detail) 属性为 false 来禁用它， 尽管如此，我们强烈建议启用它。

**注意：**直接通过 $\_COOKIE 和 setcookie() 读取和发送的 Cookie 不会被验证。

当使用 cookie 验证时，必须指定 [yii\web\Request::$cookieValidationKey](http://www.yiichina.com/doc/api/2.0/yii-web-request#$cookieValidationKey-detail)，它是用来生成上述的哈希值， 可通过在应用配置中配置 request 组件。

**return** [

'components' => [

'request' => [

'cookieValidationKey' => 'fill in a secret key here',

],

],

];

补充: [cookieValidationKey](http://www.yiichina.com/doc/api/2.0/yii-web-request#$cookieValidationKey-detail) 对你的应用安全很重要， 应只被你信任的人知晓，请不要将它放入版本控制中。

### 6、错误处理

1. [使用错误处理器](http://www.yiichina.com/doc/guide/2.0/runtime-handling-errors#using-error-handler)
2. [自定义错误显示](http://www.yiichina.com/doc/guide/2.0/runtime-handling-errors#customizing-error-display)

Yii 内置了一个[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler)错误处理器，它使错误处理更方便， Yii错误处理器做以下工作来提升错误处理效果：

* 所有非致命PHP错误（如，警告，提示）会转换成可获取异常；
* 异常和致命的PHP错误会被显示， 在调试模式会显示详细的函数调用栈和源代码行数。
* 支持使用专用的 [控制器操作](http://www.yiichina.com/doc/guide/2.0/structure-controllers#actions) 来显示错误；
* 支持不同的错误响应格式；

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler) 错误处理器默认启用， 可通过在应用的[入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)中定义常量YII\_ENABLE\_ERROR\_HANDLER来禁用。

#### 使用错误处理器

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler) 注册成一个名称为errorHandler[应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components)， 可以在应用配置中配置它类似如下：

**return** [

'components' => [

'errorHandler' => [

'maxSourceLines' => 20,

],

],

];

使用如上代码，异常页面最多显示20条源代码。

如前所述，错误处理器将所有非致命PHP错误转换成可获取异常， 也就是说可以使用如下代码处理PHP错误：

**use** **Yii**;

**use** **yii**\**base**\**ErrorException**;

**try** {

10/0;

} **catch** (ErrorException $e) {

Yii::warning("Division by zero.");

}

// execution continues...

如果你想显示一个错误页面告诉用户请求是无效的或无法处理的， 可简单地抛出一个 [HTTP exception](http://www.yiichina.com/doc/api/2.0/yii-web-httpexception)异常， 如 [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception)。 错误处理器会正确地设置响应的HTTP状态码并使用合适的错误视图页面来显示错误信息。

**use** **yii**\**web**\**NotFoundHttpException**;

**throw** **new** NotFoundHttpException();

#### 自定义错误显示

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler)错误处理器根据常量YII\_DEBUG的值来调整错误显示， 当YII\_DEBUG 为 true (表示在调试模式)， 错误处理器会显示异常以及详细的函数调用栈和源代码行数来帮助调试， 当YII\_DEBUG 为 false，只有错误信息会被显示以防止应用的敏感信息泄漏。

**信息：**如果异常是继承 [yii\base\UserException](http://www.yiichina.com/doc/api/2.0/yii-base-userexception)， 不管YII\_DEBUG为何值，函数调用栈信息都不会显示， 这是因为这种错误会被认为是用户产生的错误，开发人员不需要去修正。

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler) 错误处理器默认使用两个[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)显示错误:

* @yii/views/errorHandler/error.php: 显示不包含函数调用栈信息的错误信息是使用， 当YII\_DEBUG 为 false时，所有错误都使用该视图。
* @yii/views/errorHandler/exception.php: 显示包含函数调用栈信息的错误信息时使用。

可以配置错误处理器的 [errorView](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$errorView-detail) 和 [exceptionView](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$exceptionView-detail) 属性 使用自定义的错误显示视图。

##### 使用错误动作

使用指定的错误[操作](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 来自定义错误显示更方便， 为此，首先配置errorHandler组件的 [errorAction](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$errorAction-detail) 属性， 类似如下：

**return** [

'components' => [

'errorHandler' => [

'errorAction' => 'site/error',

],

]

];

[errorAction](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$errorAction-detail) 属性使用 [路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes)到一个操作， 上述配置表示不用显示函数调用栈信息的错误会通过执行site/error操作来显示。

可以创建 site/error 动作如下所示：

**namespace** **app**\**controllers**;

**use** **Yii**;

**use** **yii**\**web**\**Controller**;

**class** **SiteController** **extends** **Controller**

{

**public** **function** **actions**()

{

**return** [

'error' => [

'class' => 'yii\web\ErrorAction',

],

];

}

}

上述代码定义error 操作使用[yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction) 类， 该类渲染名为error视图来显示错误。

除了使用[yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction), 可定义error 动作使用类似如下的操作方法：

**public** **function** **actionError**()

{

$exception = Yii::$app->errorHandler->exception;

**if** ($exception !== **null**) {

**return** $this->render('error', ['exception' => $exception]);

}

}

现在应创建一个视图文件为views/site/error.php，在该视图文件中，如果错误动作定义为[yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction)， 可以访问该动作中定义的如下变量：

* name: 错误名称
* message: 错误信息
* exception: 更多详细信息的异常对象，如HTTP 状态码， 错误码，错误调用栈等。

**注意：**如果你使用 [基础应用模板](http://www.yiichina.com/doc/guide/2.0/start-installation) 或 [高级应用模板](http://www.yiichina.com/doc/guide/2.0/tutorial-advanced-app), 错误动作和错误视图已经定义好了。

**注意：**如果您需要在错误处理程序中重定向，请执行以下操作：

Yii::$app->getResponse()->redirect($url)->send();

**return**;

##### 自定义错误格式

错误处理器根据[响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses)设置的格式来显示错误， 如果[response format](http://www.yiichina.com/doc/api/2.0/yii-web-response#$format-detail) 响应格式为html, 会使用错误或异常视图来显示错误信息，如上一小节所述。 对于其他的响应格式，错误处理器会错误信息作为数组赋值 给[yii\web\Response::$data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail)属性，然后转换到对应的格式， 例如，如果响应格式为json，可以看到如下响应信息：

HTTP/1.1 404 Not Found

**Date**: Sun, 02 Mar 2014 05:31:43 GMT

**Server**: Apache/2.2.26 (Unix) DAV/2 PHP/5.4.20 mod\_ssl/2.2.26 OpenSSL/0.9.8y

**Transfer-Encoding**: chunked

**Content-Type**: application/json; charset=UTF-8

{

"**name**": "Not Found Exception",

"**message**": "The requested resource was not found.",

"**code**": 0,

"**status**": 404

}

可在应用配置中响应response组件的 beforeSend事件来自定义错误响应格式。

**return** [

// ...

'components' => [

'response' => [

'class' => 'yii\web\Response',

'on beforeSend' => **function** ($event) {

$response = $event->sender;

**if** ($response->data !== **null**) {

$response->data = [

'success' => $response->isSuccessful,

'data' => $response->data,

];

$response->statusCode = 200;

}

},

],

],

];

上述代码会重新格式化错误响应，类似如下：

HTTP/1.1 200 OK

**Date**: Sun, 02 Mar 2014 05:31:43 GMT

**Server**: Apache/2.2.26 (Unix) DAV/2 PHP/5.4.20 mod\_ssl/2.2.26 OpenSSL/0.9.8y

**Transfer-Encoding**: chunked

**Content-Type**: application/json; charset=UTF-8

{

"**success**": false,

"**data**": {

"**name**": "Not Found Exception",

"**message**": "The requested resource was not found.",

"**code**": 0,

"**status**": 404

}

}

### 7、日志

1. [日志消息](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-messages)
2. [日志目标](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets)
3. [性能分析](http://www.yiichina.com/doc/guide/2.0/runtime-logging#performance-profiling)

Yii提供了一个强大的日志框架，这个框架具有高度的可定制性和可扩展性。使用这个框架， 你可以轻松地记录各种类型的消息，过滤它们， 并且将它们收集到不同的目标，诸如文件，数据库，邮件。

使用Yii日志框架涉及下面的几个步骤：

* 在你代码里的各个地方记录 [log messages](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-messages)；
* 在应用配置里通过配置 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 来过滤和导出日志消息；
* 检查由不同的目标导出的已过滤的日志消息（例如：[Yii debugger](http://www.yiichina.com/doc/guide/2.0/tool-debugger)）。

在这部分，我们主要描述前两个步骤。

#### 日志消息

记录日志消息就跟调用下面的日志方法一样简单：

* Yii::trace()：记录一条消息去跟踪一段代码是怎样运行的。这主要在开发的时候使用。
* Yii::info()：记录一条消息来传达一些有用的信息。
* Yii::warning()：记录一个警告消息用来指示一些已经发生的意外。
* Yii::error()：记录一个致命的错误，这个错误应该尽快被检查。

这些日志记录方法针对 严重程度 和 类别 来记录日志消息。 它们共享相同的函数签名 function ($message, $category = 'application')，$message代表要被 记录的日志消息，而 $category 是日志消息的类别。在下面的示例代码中，在默认的类别 application 下 记录了一条跟踪消息：

Yii::trace('start calculating average revenue');

**注意：**日志消息可以是字符串，也可以是复杂的数据，诸如数组或者对象。 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 的义务是正确处理日志消息。默认情况下， 假如一条日志消息不是一个字符串，它将被导出为一个字符串，通过调用 yii\helpers\VarDumper::export()。

为了更好地组织和过滤日志消息，我们建议您为每个日志消息指定一个适当的类别。您可以为类别选择一个分层命名方案， 这将使得 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 在基于它们的分类来过滤消息变得更加容易。 一个简单而高效的命名方案是使用PHP魔术常量 \_\_METHOD\_\_ 作为分类名称。 这种方式也在Yii框架的核心代码中得到应用， 例如，

Yii::trace('start calculating average revenue', **\_\_METHOD\_\_**);

\_\_METHOD\_\_ 常量计算值作为该常量出现的地方的方法名（完全限定的类名前缀）。 例如，假如上面那行代码在这个方法内被调用，则它将等于字符串 'app\controllers\RevenueController::calculate'。

**注意：**上面所描述的日志方法实际上是 [logger object](http://www.yiichina.com/doc/api/2.0/yii-log-logger) 对象（一个通过表达式 Yii::getLogger() 可访问的单例） 的方法 [log()](http://www.yiichina.com/doc/api/2.0/yii-log-logger#log()-detail) 的一个快捷方式。当足够的消息被记录或者当应用结束时， 日志对象将会调用一个 [message dispatcher](http://www.yiichina.com/doc/api/2.0/yii-log-dispatcher) 调度对象将已经记录的日志消息发送到已注册的 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 目标中。

#### 日志目标

一个日志目标是一个 [yii\log\Target](http://www.yiichina.com/doc/api/2.0/yii-log-target) 类或者它的子类的实例。 它将通过他们的严重层级和类别来过滤日志消息，然后将它们导出到一些媒介中。 例如，一个 [database target](http://www.yiichina.com/doc/api/2.0/yii-log-dbtarget) 目标导出已经过滤的日志消息到一个数据的表里面， 而一个 [email target](http://www.yiichina.com/doc/api/2.0/yii-log-emailtarget)目标将日志消息导出到指定的邮箱地址里。

在一个应用里，通过配置在应用配置里的 log [application component](http://www.yiichina.com/doc/guide/2.0/structure-application-components) ，你可以注册多个日志目标。 就像下面这样：

**return** [

// the "log" component must be loaded during bootstrapping time

'bootstrap' => ['log'],

'components' => [

'log' => [

'targets' => [

[

'class' => 'yii\log\DbTarget',

'levels' => ['error', 'warning'],

],

[

'class' => 'yii\log\EmailTarget',

'levels' => ['error'],

'categories' => ['yii\db\\*'],

'message' => [

'from' => ['log@example.com'],

'to' => ['admin@example.com', 'developer@example.com'],

'subject' => 'Database errors at example.com',

],

],

],

],

],

];

**注意：**log 组件必须在 [bootstrapping](http://www.yiichina.com/doc/guide/2.0/runtime-bootstrapping) 期间就被加载，以便于它能够及时调度日志消息到目标里。 这是为什么在上面的代码中，它被列在 bootstrap 数组中的原因。

在上面的代码中，在 [yii\log\Dispatcher::$targets](http://www.yiichina.com/doc/api/2.0/yii-log-dispatcher#$targets-detail) 属性里有两个日志目标被注册：

* 第一个目标选择的是错误和警告层级的消息，并且在数据库表里保存他们；
* 第二个目标选择的是错误层级的消息并且是在以 yii\db\ 开头的分类下，并且在一个邮件里将它们发送到 admin@example.com 和 developer@example.com。

Yii配备了以下的内建日志目标。请参考关于这些类的API文档， 并且学习怎样配置和使用他们。

* [yii\log\DbTarget](http://www.yiichina.com/doc/api/2.0/yii-log-dbtarget)：在数据库表里存储日志消息。
* [yii\log\EmailTarget](http://www.yiichina.com/doc/api/2.0/yii-log-emailtarget)：发送日志消息到预先指定的邮箱地址。
* [yii\log\FileTarget](http://www.yiichina.com/doc/api/2.0/yii-log-filetarget)：保存日志消息到文件中.
* [yii\log\SyslogTarget](http://www.yiichina.com/doc/api/2.0/yii-log-syslogtarget)：通过调用PHP函数 syslog() 将日志消息保存到系统日志里。

下面，我们将描述所有日志目标的公共特性。

##### 消息过滤

对于每一个日志目标，你可以配置它的 yii\log\Target::levels 和 [categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性来指定哪个消息的严重程度和分类目标应该处理。

yii\log\Target::levels 属性是由一个或者若干个以下值组成的数组：

* error：相应的消息通过 Yii::error() 被记录。
* warning：相应的消息通过 Yii::warning() 被记录。
* info：相应的消息通过 Yii::info() 被记录。
* trace：相应的消息通过 Yii::trace() 被记录。
* profile：相应的消息通过 Yii::beginProfile() 和 Yii::endProfile() 被记录。更多细节将在 [Profiling](http://www.yiichina.com/doc/guide/2.0/runtime-logging#performance-profiling) 分段解释。

如果你没有指定 yii\log\Target::levels 的属性， 那就意味着目标将处理 任何 严重程度的消息。

[categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性是一个包含消息分类名称或者模式的数组。 一个目标将只处理那些在这个数组中能够找到对应的分类或者其中一个相匹配的模式的消息。 一个分类模式是一个以星号 \* 结尾的分类名前缀。假如一个分类名与分类模式具有相同的前缀， 那么该分类名将和分类模式相匹配。例如， yii\db\Command::execute 和 yii\db\Command::query都是作为分类名称运用在 [yii\db\Command](http://www.yiichina.com/doc/api/2.0/yii-db-command) 类来记录日志消息的。 它们都是匹配模式 yii\db\\*。

假如你没有指定 [categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性， 这意味着目标将会处理 任何 分类的消息。

除了通过 [categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性设置白名单分类，你也可以通过 [except](http://www.yiichina.com/doc/api/2.0/yii-log-target#$except-detail) 属性来设置某些分类作为黑名单。假如一条消息的分类在这个属性中被发现或者是匹配其中一个， 那么它将不会在目标中被处理。

在下面的目标配置中指明了目标应该只处理错误和警告消息，当分类的名称匹配 yii\db\\* 或者是 yii\web\HttpException:\* 的时候， 但是除了 yii\web\HttpException:404。

[

'class' => 'yii\log\FileTarget',

'levels' => ['error', 'warning'],

'categories' => [

'yii\db\\*',

'yii\web\HttpException:\*',

],

'except' => [

'yii\web\HttpException:404',

],

]

**注意：**当一个HTTP异常通过 [error handler](http://www.yiichina.com/doc/guide/2.0/runtime-handling-errors) 被捕获的时候，一个错误消息将以 yii\web\HttpException:ErrorCode 这样的格式的分类名被记录下来。例如，[yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception)将会引发一个分类是 yii\web\HttpException:404 的 错误消息。

##### 消息格式化

日志目标以某种格式导出过滤过的日志消息。例如， 假如你安装一个 [yii\log\FileTarget](http://www.yiichina.com/doc/api/2.0/yii-log-filetarget) 类的日志目标， 你应该能找出一个日志消息类似下面的 runtime/log/app.log 文件：

2014-10-04 18:10:15 [::1][][-][trace][yii\base\Module::getModule] Loading module: debug

默认情况下，日志消息将被格式化，格式化的方式遵循 [yii\log\Target::formatMessage()](http://www.yiichina.com/doc/api/2.0/yii-log-target#formatMessage()-detail)：

Timestamp [IP address][User ID][Session ID][Severity Level][Category] Message Text

你可以通过配置 [yii\log\Target::$prefix](http://www.yiichina.com/doc/api/2.0/yii-log-target#$prefix-detail) 的属性来自定义格式，这个属性是一个PHP可调用体返回的自定义消息前缀。 例如，下面的代码配置了一个日志目标的前缀是每个日志消息中当前用户的ID(IP地址和Session ID被删除是由于隐私的原因)。

[

'class' => 'yii\log\FileTarget',

'prefix' => **function** ($message) {

$user = Yii::$app->has('user', **true**) ? Yii::$app->get('user') : **null**;

$userID = $user ? $user->getId(**false**) : '-';

**return** "[$userID]";

}

]

除了消息前缀以外，日志目标也可以追加一些上下文信息到每组日志消息中。 默认情况下，这些全局的PHP变量的值被包含在：$\_GET, $\_POST, $\_FILES, $\_COOKIE,$\_SESSION 和 $\_SERVER 中。 你可以通过配置 [yii\log\Target::$logVars](http://www.yiichina.com/doc/api/2.0/yii-log-target#$logVars-detail) 属性适应这个行为， 这个属性是你想要通过日志目标包含的全局变量名称。 举个例子，下面的日志目标配置指明了只有 $\_SERVER 变量的值将被追加到日志消息中。

[

'class' => 'yii\log\FileTarget',

'logVars' => ['\_SERVER'],

]

你可以将 logVars 配置成一个空数组来完全禁止上下文信息包含。 或者假如你想要实现你自己提供上下文信息的方式， 你可以重写 [yii\log\Target::getContextMessage()](http://www.yiichina.com/doc/api/2.0/yii-log-target#getContextMessage()-detail) 方法。

##### 消息跟踪级别

在开发的时候，通常希望看到每个日志消息来自哪里。这个是能够被实现的，通过配置 log 组件的 yii\log\Dispatcher::traceLevel 属性， 就像下面这样：

**return** [

'bootstrap' => ['log'],

'components' => [

'log' => [

'traceLevel' => YII\_DEBUG ? 3 : 0,

'targets' => [...],

],

],

];

上面的应用配置设置了 yii\log\Dispatcher::traceLevel 的层级，假如 YII\_DEBUG 开启则是3，否则是0。 这意味着，假如 YII\_DEBUG 开启，每个日志消息在日志消息被记录的时候， 将被追加最多3个调用堆栈层级；假如 YII\_DEBUG 关闭， 那么将没有调用堆栈信息被包含。

**注意：**获得调用堆栈信息并不是不重要。因此， 你应该只在开发或者调试一个应用的时候使用这个特性。

##### 消息刷新和导出

如上所述，通过 [logger object](http://www.yiichina.com/doc/api/2.0/yii-log-logger) 对象，日志消息被保存在一个数组里。 为了这个数组的内存消耗，当数组积累了一定数量的日志消息， 日志对象每次都将刷新被记录的消息到 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 中。 你可以通过配置 log 组件的 yii\log\Dispatcher::flushInterval 属性来自定义数量：

**return** [

'bootstrap' => ['log'],

'components' => [

'log' => [

'flushInterval' => 100, // default is 1000

'targets' => [...],

],

],

];

**注意：**当应用结束的时候，消息刷新也会发生，这样才能确保日志目标能够接收完整的日志消息。

当 [logger object](http://www.yiichina.com/doc/api/2.0/yii-log-logger) 对象刷新日志消息到 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 的时候，它们并 不能立即获取导出的消息。相反，消息导出仅仅在一个日志目标累积了一定数量的过滤消息的时候才会发生。你可以通过配置 个别的 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 的 [exportInterval](http://www.yiichina.com/doc/api/2.0/yii-log-target#$exportInterval-detail) 属性来 自定义这个数量，就像下面这样：

[

'class' => 'yii\log\FileTarget',

'exportInterval' => 100, // default is 1000

]

因为刷新和导出层级的设置，默认情况下，当你调用 Yii::trace() 或者任何其他的记录方法，你将不能在日志目标中立即看到日志消息。 这对于一些长期运行的控制台应用来说可能是一个问题。为了让每个日志消息在日志目标中能够立即出现， 你应该设置 yii\log\Dispatcher::flushInterval 和 [exportInterval](http://www.yiichina.com/doc/api/2.0/yii-log-target#$exportInterval-detail) 都为1， 就像下面这样：

**return** [

'bootstrap' => ['log'],

'components' => [

'log' => [

'flushInterval' => 1,

'targets' => [

[

'class' => 'yii\log\FileTarget',

'exportInterval' => 1,

],

],

],

],

];

**注意：**频繁的消息刷新和导出将降低你到应用性能。

##### 切换日志目标

你可以通过配置 [enabled](http://www.yiichina.com/doc/api/2.0/yii-log-target#$enabled-detail) 属性来开启或者禁用日志目标。 你可以通过日志目标配置去做，或者是在你的代码中放入下面的PHP申明：

Yii::$app->log->targets['file']->enabled = **false**;

上面的代码要求您将目标命名为 file，像下面展示的那样， 在 targets 数组中使用使用字符串键：

**return** [

'bootstrap' => ['log'],

'components' => [

'log' => [

'targets' => [

'file' => [

'class' => 'yii\log\FileTarget',

],

'db' => [

'class' => 'yii\log\DbTarget',

],

],

],

],

];

##### 创建新的目标

创建一个新的日志目标类非常地简单。你主要需要实现 [yii\log\Target::export()](http://www.yiichina.com/doc/api/2.0/yii-log-target#export()-detail) 方法来发送 [yii\log\Target::$messages](http://www.yiichina.com/doc/api/2.0/yii-log-target#$messages-detail) 数组的 内容到一个指定的媒体中。你可以调用 [yii\log\Target::formatMessage()](http://www.yiichina.com/doc/api/2.0/yii-log-target#formatMessage()-detail) 方法去格式化每个消息。 更多细节，你可以参考任何一个包含在Yii发布版中的日志目标类。

#### 性能分析

性能分析是一个特殊的消息记录类型，它通常用在测量某段代码块的时间， 并且找出性能瓶颈是什么。举个例子，[yii\db\Command](http://www.yiichina.com/doc/api/2.0/yii-db-command) 类 使用性能分析找出每个数据库查询的时间。

为了使用性能分析，首先确定需要进行分析的代码块。 然后像下面这样围住每个代码块：

\Yii::beginProfile('myBenchmark');

...code block being profiled...

\Yii::endProfile('myBenchmark');

这里的 myBenchmark 代表一个唯一标记来标识一个代码块。之后当你检查分析结果的时候， 你将使用这个标记来定位对应的代码块所花费的时间。

对于确保 beginProfile 和 endProfile 对能够正确地嵌套，这是很重要的。 例如，

\Yii::beginProfile('block1');

// some code to be profiled

\Yii::beginProfile('block2');

// some other code to be profiled

\Yii::endProfile('block2');

\Yii::endProfile('block1');

假如你漏掉 \Yii::endProfile('block1') 或者切换了 \Yii::endProfile('block1') 和 \Yii::endProfile('block2') 的 顺序，那么性能分析将不会工作。

对于每个被分析的代码块，一个带有严重程度 profile 的日志消息被记录。 你可以配置一个 [log target](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 去收集这些 消息，并且导出他们。[Yii debugger](http://www.yiichina.com/doc/guide/2.0/tool-debugger) 有一个内建的性能分析面板能够展示分析结果。

**数据库连接**：

想要访问数据库，首先要创建一个yii\db\Connection实例来与数据库连接。

Exp:1 用Yii数据库连接类连接数据库

$db = **new** yii\db\Connection([

'dsn' => 'mysql:host=localhost;dbname=example',

'username' => 'root',

'password' => '',

'charset' => 'utf8',

]);

由于数据库连接经常需要在多个地方使用，以应用组件的方式将其配置如下：

Exp:2 配置数据库组件

Return[

‘Components’ => [

‘db’ => [

‘class’=>’Yii\db\Connection’, //数据库连接类

‘dsn’=>’mysql:host=127.0.0.1;dbname=mydbname’,//数据库地址，名称

‘username’=>’root’, //用户名

‘password’=>’root’, //密码

‘charset’=>’UTF8’, //使用字符集

‘enableSchemaCache’=>’true’, //开启数据库缓存

‘schemaCache’=>’cache’, //缓存到runtime\cache目录

‘schemaCachDruation’=>3600 //缓存时间3600秒

],

],

];

|  |
| --- |
| 注：如果需要访问多个数据库， 可以配置多个DB应用组件。DSN格式数据库对照：  MySQL, MariaDB: mysql:host=localhost;dbname=mydatabase  SQLite: sqlite:/path/to/database/file  PostgreSQL: pgsql:host=localhost;port=5432;dbname=mydatabase  CUBRID: cubrid:dbname=demodb;host=localhost;port=33000  MS SQL Server (via sqlsrv driver): sqlsrv:Server=localhost;Database=mydatabase  MS SQL Server (via dblib driver): dblib:host=localhost;dbname=mydatabase  MS SQL Server (via mssql driver): mssql:host=localhost;dbname=mydatabase  Oracle: oci:dbname=//localhost:1521/mydatabase  ODBC数据库，应该配置yii\db\Connection::driverName属性，以便yii知道实际的数据库类。如：  'db' => [  'class' => 'yii\db\Connection',  'driverName' => 'mysql',  'dsn' => 'odbc:Driver={MySQL};Server=localhost;Database=test',  'username' => 'root',  'password' => '',  ], |

Exp:3 连接数据库

Yii::$app->db;

Public function actionIndex(){

$connect = Yii::$app->db;

}

**初始化数据库环境变量**

**信息：**当你实例化一个 DB Connection 时， 直到你第一次执行 SQL 或者你明确地调用 [open()](http://www.yiichina.com/doc/api/2.0/yii-db-connection#open()-detail) 方法时， 才建立起实际的数据库连接。

**提示：**有时你可能想要在建立起数据库连接时立即执行一些语句来初始化一些环境变量 (比如设置时区或者字符集), 你可以通过为数据库连接的 [afterOpen](http://www.yiichina.com/doc/api/2.0/yii-db-connection#EVENT_AFTER_OPEN-detail) 事件注册一个事件处理器来达到目的。 你可以像这样直接在应用配置中注册处理器：

'db' => [

// ...

'on afterOpen' => **function**($event) {

// $event->sender refers to the DB connection

$event->sender->createCommand("SET time\_zone = 'UTC'")->execute();

}

],

**数据库操作**

DAO数据库操作对象，在查询中必须输入原生的SQL语句。

数据库操作\_查询

DAO ⬄databases Access Object

1 queryAll（） //查询满足条件的所有记录

2 queryOne（） //按条件查询一条记录，如果有多条记录满足则取第一条

3 queryScalar（） //查询统计，返回一个数值

4 queryColumn（） //查询满足条件的某一列，返回一个一维数组

Exp:1 查询满足条件的所有记录

Public function actionIndex(){

$conn = Yii::$app->db;

$sql = ‘select \* from user’;

$command = $conn->createCommand($sql);

$data = $command->queryAll();

Var\_dump($data);

}

Exp:2

Public function actionIndex(){

$sql = ‘select \* from user’;

$data = Yii::$app->db->creatCommand($sql)->queryAll();

Var\_dump($data);

}

Exp:3按条件查询一条记录，如果有多条记录满足则取第一条

Public function actionIndex(){

$sql = ‘select \* from user where id=8’;

$data = Yii::$app->db->creatComman($sql)->queryOne();

Var\_dump($data);

}

Exp:4查询统计，返回一个数值

Public function actionIndex(){

$sql = ‘select count(\*) as total from user ‘;

$data = Yii::$app->db->creatComman($sql)->queryScalar();

Echo $data;

}

Exp5: queryColumn();返回一个一维数组

Public function actionIndex(){

$sql = ‘select username from user ‘;

$data = Yii::$app->db->creatComman($sql)->queryColumn();

Var\_dump($data);

}

数据库操作\_插入

Exp1:

Public function actionIndex(){

$sql = ‘inert into user (username) values (“ab”)’;

$res = Yii::$app->db->createCommand($sql)->execute();

Echo $res; //返回的受影响记录的条数

}

Exp2:

Public function actionIndex(){

$result = Yii::$app->db->createCommand()->insert(‘user’,[

‘username’=>’ab’

])->execute();

Echo $result;

}

Exp3:批量插入数据

Public function actionIndex(){

$result = Yii:: $app->createCommand()->batchInsert(

‘user’, //表名

[‘username’], //字段名

[[‘ab1’],[‘ab2’]] //字段值

)->execute();

Echo $result; //受影响记录的条数

}

数据库操作\_更新

Exp1:仿上例exp1(略)

Public funciton actionIndex(){

$result = Yii::$app ->db->createCommand()->updata(

‘user’, //表名

[‘username’=>’zhangsan’], //更新字段

[‘id’=>1] //更新条件

)->execute();

Echo $result; //受影响记录的条数

}

注：如果不写名更新条件将更新对应的所有字段

数据库操作\_删除

Exp1:仿上例exp1(略)

Public function actionIndex(){

$result = Yii::$app->db->createCommand()->delete(

‘user’, //表名

[‘id’=>1] //删除条件

)->execute();

Echo $result;

}

注：如果不写名删除条件将删除所有数据

参数绑定

有三种方式：

bindValue|bindValues|bindParam

exp1:bindValue绑定一个值

public function actionIndex(){

$conn = Yii::$app->db;

$command = $conn->createCommand(“select \* from user where id=:id”);

$command->bindValue(‘:id’,25);

$command->bindValue(‘username’,’zhangsan’);

$data = $command -> queryOne();

Vau\_dump($data);

}

注：参数绑定可以有效地防止SQL攻击。

Exp2:bindValues 绑定多个值

Public function actionIndex(){

$conn = Yii::$app->db;

$command = $conn->createCommand(“select \* from `user` where id = :id and username = :username”);

$command->bindValues([

‘:id’ => 25,

‘:username’ => ‘zhangsan’

]);

$data = $command –> queryOne();

Var\_dump($data);

}

Exp3:bindParam绑定参数(其中参数是变量的引用)

Public function actionIndex(){

$id = 25;

$conn = Yii::$app->db;

$command = $conn->createCommand(“selsect \* from user where id = :id);

$command -> bindParam(“:id”,$id);

$data0 = $command -> queryOne(); //这将查询到id=25的记录

注：如果我们继续给id赋值

$id = 24;

$data1 = $command -> queryOne(); //这将查询到id=24的记录

}

QueryBuilder数据库操作查询构造器

需要引入：use yii\db\QueryBuilder;

use yii\db\Query;

Exp1:

Public functon actionIndex(){

$query = new Query();

$query -> select([‘username’]);

$query-> from(‘user’);

Var\_dump($query->all());

}

注：1、构造sql语句的时候，在$query->all()|$query->one()|…执行之前并没有组装sql语句。

2、构造sql语句的时候可以通过$query->createCommand()让构造器组装sql语句。

3、通过$query->createCommand()->sql查看组装后的sql语句。

QueryBuilder 中的 select查询

Public function actionIndex(){

$query = new Query();

$query -> from(‘user’);

$query -> select(

[‘identity\_card’=>”CONCAT(id,’-‘,username)”]

);

//$query->sellect([‘username’])->distinct(); //去掉重复

Var\_dump($query->all());

}

注：在Yiisql查询中的，当表名或字段名用一个数组表示时，键名为相应的别名，键值为其自身。

QueryBuilder 中的where查询

基本结构如下：

Public function actionIndex(){

$query = new Query();

$query->from([‘user’]);

$query->select();

$query->where();

//Echo $query->createCommand()->sql; //查看构造的sql语句

Echo “<br/>”;

Var\_dump($query->all());

}

注

关于$query->where()的用法：

1. 可以接收字符串，如：$query->where(‘id=24’);推荐将其参数化，如下：
2. $query->where(‘id=:id’)->addParams([‘:id’=>24]); 或 $query->where(‘id=:id’,[‘:id’=>24]);
3. 可以接收数组，简单查询如：

$query->where([

‘username’=>’ab’,

‘id’=>[24,29] //id in(24,29)

])

这种简单类型的查询，数组的键名表示查询的字段，键值表示匹配的值

1. 可以接收数组，复杂查询如：

$query->where([‘<>’,’id’,24]); //表示id不等于24

$query->where([‘in’,’id’,[24,25]]); //表示id等于24或等于25

$query->where([‘like’,’username’,’ab’]);

关于$query->orderBy()的用法：

$query->orderBy(‘id desc’); 推荐使用以下方式

$query->orderBy([

‘id’=>SORT\_DESC, //id降序排

‘username’=>SORT\_ASC //username升序排

]);

QueryBuilder 中的 limit查询

Exp1:

Public function actionIndex(){

$query = new Query();

$query->from(‘user’);

$query->orderBy([‘id’=>SORT\_ASC]);

$query->limit(4); //默认从第0条记录开始取4条

Echo $query->createCommand()->sql;

Var\_dump($query->all());

}

QueryBuilder 中的 offset查询

Exp1:

$query->offset(2); //表示偏移量

注：offset()常与limit()配合使用，接合上两例表示：从第2条开始取4条记录

QueryBuilder 中的 groupby查询

用于将表中字段进行分组后统计.

Exp1:查询某个省有多少个会员

$query->select([

‘province’,

‘user\_total’=>”count(\*)”

]);

$query->groupBy([

‘province’

]);

QueryBuilder 中的 having查询

按条件将字段分组后进行统计，它具有having查询自身的特点

Exp1:统计会员数大于2的省份

$query->select([

‘province’,

‘user\_total’=>”count(\*)”

]);

$query->groupBy([‘province’]);

$query->having([‘>’,’user\_total’,1]);

**数据提供者**ActiveDataProvider

引入的类：yii\data\ActiveDataProvider;

分页实例：

Controller文件

<?php

Namespace app\controllers;

Use yii\web\Controller;

Use yii;

Use yii\db\Query;

Use yii\data\ActiveDataProvider;

Class Course112Controller extends Controller{

/\*\*

\*使用query Builder和ActiveCataProvider来显示数据

\*/

Public function actionIndex(){

$query = new Query();

$query -> from(‘user’);

$query->select(['序号'=>'id','用户名'=>'Username','邮箱'=>'Email']); //可指定字段及表头名称

$dataProvider = new ActiveDataProvider([

‘query’ => $query,

‘pagination’ => [

‘pageSize’ => 5

]

]);

Return $this -> render(‘index’,[

‘dataProvider’ => $dataProvider

]);

}

}

Viewer 文件

<?php

Use yii\grid\GridView;

use yii\base\Widget;

?>

<?php

Echo GridView::widget([‘dataProvider’=>$ dataProvider’]);

?>

ActiveRecord数据库操作活动记录（简称AR）

说明：

1. 一个AR类对应数据库的一张表
2. AR类的一个对象对应表中的一条记录
3. AR类的一个属性对就表中列的属性
4. 新建一个对象的时候，如果你去更新它，AR就去同步更新这张表

AR与model

Exp1-1:新建一个user表的AR类

<?php

Namespace app\models; //命名空间统一在app

Use yii\db\ActiveRecord;

Class User extends ActiveRecord{

/\*\*

\*该函数用于指定表与类的对应关系

\*/

Public static function tableName(){

Return “user”;

}

}

?>

AR与控制器

Exp1-2: 控制器中使用AR类

<?php

Namespace app\controllers;

Use yii\web\Controller;

Use app\models\User;

Class Course113Controller extends Controller{

Public function actionIndex(){

Var\_dump(User::find()->limit(5)->all());

}

}

AR与gii

Exp2-1:通过gii生成model类

<?php

Namespace app\models;

Use yii\db\ActiveRecord;

Class User extends ActiveRecord{

/\*\*

\*该函数用于指定表与类的对应关系

\*/

Public static function tableName(){

Return “user”;

}

/\*\*

\*说明：使用AR向数据库写入数据，需要通过rules()检查。如果插入数据失败，多数是因为检查没通过

\*可在规则中加入message字段定义错误信息输出内容如：[[‘username’,’province’,’message’=>’这是错误提示信息内容’],’string’,’max’=>32]

\*/

Public function rules(){  
 return[

[[‘username’,’province’],’string’,’max’=>32],

[‘province’,’integer’],

[‘username’,function($attr,$params){ //自定义规则exp1:通过匿名函数定义

If($this->$attr == ‘ab’){

Return true;

}else{

$this->addError(‘username’,’你不是ab,不能通过’);

}

}],

[‘username’,’myrules0’] //自定义规则exp2:使用自定义规则

];

}

/\*\*

\*自定义规则exp2：

\*/

Public function myrules0’($attr,$params){

If($this->$attr == ‘ab’){

Return true;

}else{

$this->addError(‘username’,’你不是ab,不能通过’);

}

}

/\*\*

\* 1、在该方法中可以设置字段名在网页的显示名称

\* 2、在控制中通过 (new User())->getAttributeLabel(“username”);设置的字段显示名称

\*/

Public function attributeLabels(){

Return [

‘id’=>’ID’,

‘username’=>’用户名’

]

}

}

?>

Exp2-1:插入数据

<?php

Namespace app\controllers;

Use yii\web\Controller;

User app\odels\User;

Class Course115Controller extends Controller{

Public function actionIndex(){

$model = new User();

$model->username = ‘a’;

$model->province = ‘b’;

If($model->save() == false){; //如果数据存在则更新，不存在则插入。

Var\_dump($model->getErrors()); //如果操作失败则输出错误信息

}

}

}

?>

AR与自定义验证规则

详解活动记录模型model中的自定义验证规则

除系统自带验证规则，我们可以根据需要自定义验证规则。如下所述：

自定义规则exp1:通过匿名函数定义

Public function rules(){  
 return[

[‘username’,function($attr,$params){

If($this->$attr == ‘ab’){

Return true;

}else{

$this->addError(‘username’,’你不是ab,不能通过’);

}

}],

];

}

自定义规则exp2-1：在当当前类中通过公共方法定义

Public function myrules0’($attr,$params){

If($this->$attr == ‘ab’){

Return true;

}else{

$this->addError(‘username’,’你不是ab,不能通过’);

}

}

exp2-2: 使用自定义规则

Public function rules(){

Return[

[‘username’,’myrules0’] //自定义规则exp2:使用自定义规则

];

}

验证规则总结：

上述通过匿名函数和公共方法都可以自定义一个验证规则，但代码复用性较差。如果一个验证方法会在系统中很多地方都用到，那么可以在应用根目录下建一验证规则目录（myrules），存放验证类。

自定义验证规则

exp3-1：自定义验证类

<?php

Namespace app\myrules;

Use yii\validators\Validator; //继承系统验证类（必须）

Class Myrules extends Validator{

/\*\*

\*说明：

\*重载validateAttribute()

\* $model object 被验证的模型

\* $attribute string被验证的属性

\*/

Public function validateAttribute($model,$attribute){

If($model -> $attribute == ‘ab’){

Return true;

}else{

$ths->addError($model, $attribute,’你不是ab,不能通过’);

}

}

}

exp3-2：使用自定义验证类

<?php

Namespace app\models;

Use yii\db\ActiveRecord;

Use app\myrules\Myrules; //引入自定义验证类

Class User extends ActiveRecord{

Public function rules(){  
 return[

[‘username’, Myrules::className()] //自定义规则exp2:使用自定义规则

];

}

｝

ActiveRecord AR 操作数据库

详解 yii\db\ActiveRecord::save(); 该方法源于Interface yii\db\ActiveRecordInterface

如果是一个新的对象则等价于yii\db\ActiveRecord::insert()

否则等价于yii\db\ActiveRecord::[update()](http://www.yiichina.com/doc/api/2.0/yii-db-activerecordinterface#update()-detail)

要求：引入 use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Exp1-1:

<?php

Namespace app\controllers;

use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Class Course117Controller extends Controller{

Public function actionIndex(){

$model = new User(); //这是一个新的对象

$model -> username = ‘a’;

$model -> province = ‘c’;

$model -> save();

}

}

注： yii\db\ActiveRecord::attributes = [];批量赋值如：

$model -> attributes = [

‘username’ => ‘a’,

‘province’ => ‘gx’

];

其中的每一个字段都必须在rules()中有相应字段的验证规则。否则数据不能通过。

如果启用了[‘username’,’safe’]规则，将不再对username字段进行验证。Rules只限制批量赋值。

Exp1-2:

Public function actionIndex(){

$model = User::findOne(24); //查找已经存在的对象（针对主键）

$model->username = ‘aa’;

$model->save(); //这等价于update()

}

ActiveRecord AR 查询语句

要求：引入 use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Exp1:

<?php

Namespace app\controllers;

use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Class Course117Controller extends Controller{

Public function actionIndex(){

$data = User::find() -> where([‘id’=>24]) -> all();

Var\_dump($data);

}

}

?>

Exp2:

Public function actionIndex(){

$data = User::find() -> limit(2) -> all();

Var\_dump();

}

提示：在活动记录中查询语句的结构和查询构造器是一至的。

注：在上述的查询中以对象结果集返回。在返回的对象中由查询结果集和对象其它属性组成。如果处理海量数据，这些属性将很大程度影响性能。我们只需要得到查询结果，可以使用asArray()方法获取 。

Exp1:

Public function actionIndex(){

$data = User::find() -> asArray() -> all();

Var\_dump($data);

}

ActiveRecord AR 删除语句

要求：引入 use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Exp1:

<?php

Namespace app\controllers;

use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Class Course117Controller extends Controller{

Public function actionIndex(){

User::deleteAll([‘in’,’id’,[24,25]]);

}

}

ActiveRecord AR 更新语句

要求：引入 use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Exp1:

<?php

Namespace app\controllers;

use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Class Course117Controller extends Controller{

Public function actionIndex(){

User::updateAll([‘username’=>’ab’]);

}

}

ActiveRecord AR 连表查询

应用场景 ：

User表

Id username province team\_id

10414 abei1 河南省 1

16415 abai 河北省 1

16416 王浩 北京市 2

Team表

Id name

1 A组

2 B组

Exp1:查询abei1属性与组的组名(1对1关系)

<?php

Namespace app\controllers;

use yii\web\Controller; //控制器

Use yii\models\User; //活动记录类

Class Course117Controller extends Controller{

Public function actionIndex(){

$model = User::findOne(16414);

Var\_dump($model -> team); //访问team表就像访问user表的一个属性

Echo $model->team->name;

}

}

注：1、在活动记录中以hasOne()表示1对1的连表查询

2、必须在user表对象的类中定义表与表连接之间的关联关系(用getter()方法表示)，如：

Public function getTeam(){

//team表的id = user表的team\_id

Return $this->hasOne(Team::className(),[‘id’=>’team\_id’]);

}

Exp2:查询A组所有成员（1对多）

<?php

Namespace app\controllers;

use yii\web\Controller; //控制器

Use yii\models\Team; //活动记录类

Class Course117Controller extends Controller{

Public function actionIndex(){

$model = Team::findOne(1);

Var\_dump($model -> user); //访问team表就像访问user表的一个属性

Echo $model->user->username;

}

}

注：1、在活动记录中以hasMany()表示1对多的连表查询

2、必须在team表对象的类中定义表与表连接之间的关联关系(用getter()方法表示)，如：

Public function getUser(){

//user表的team\_id = team表的di

Return $this->hasMany(User::className(),[‘team\_id’=>’ id’]);

}

3、hasMany(),hasOne()返回的是一个query对象，详述如下：

在这里Query类的所有方法都可用

Return $this->hasMany(User::className(),[‘team\_id’=>’ id’]) ->count(); //统计了有多少人

Return $this->hasMany(User::className(),[‘team\_id’=>’ id’]) -> where([‘id’=>16414]); //只返回id=6414的记录

其实hasMany(User::className(),[‘team\_id’=>’ id’])已经返回user的结果集，后面的子查询是对该结果集的

进一步操作。同理又如：

Return $this->hasMany(User::className(),[‘team\_id’=>’ id’]) -> orderBy([‘id’=>SORT\_DESC]); //排序后返回

Return $this->hasMany(User::className(),[‘team\_id’=>’ id’]) -> limit(1);

AR与自定义Model的关系

在上述的model中诸例，model继承了AR类以实现操作表的能力。在实际开发中，很多model不需要继承AR,我们可以自定义这样的model来处理逻辑业务，这样的model我们需要让它继承yii\base\Model。

总述:无论是自定义model类还是继承了AR类的model类，它们都继承了yii\base\Model。

验证中的场景

一、设置场景

Rules[]的使用，本身是分场景的。不同场景下所验证的内容不一样。如loginForm model在登录和注册所验证的字段内容是不一样的。

Exp1\_1:注册时所用字段和登录时用字段

Public function rules(){

Return[

[[‘username’,’password’,’province’],’required]

]

}

Exp1\_2:登录时用字段

Public function rules(){

Return[

[[‘username’,’password’],’required]

]

}

为了将两个类似的规则写到一个model中，需要将它们分别对待。于是引入了场景的概念secnarios()

重写scenarios()我们可以让系统在注册时使用Exp1\_1中的验证规则，在登录时使用Exp1\_2中的验证规则。

场景的设置我们可以通过两种方式实现：

方式一：声明使用场景

Public function rules(){

Return[

[[‘username’,’password’,’province’],’require’,’on’=>’reg’], //reg场景启用该验证规则

[[‘username’,’password’],’require’,’on’=>’login’], //login场景启用该验证规则

];

}

方式二：重载场景方法

public function scenarios(){

return[

‘login’ => [‘username’,’password’],

‘reg’=>[‘username’,password’,’province’]

];

}

注：当我们重载场景方法时，如果不设置default场景，那么会因default场景不存在而导致系统报错。

二、使用场景

在控制器中使用model中设置的场景，需要在操作下进行指定 ：

Exp1:

<?php

Namespace app\controllers;

Use app\models\Loginfrom;

Use yii\web\Controller;

Class Course119Controller extends Controller{

Public function actionIndex(){

$model = new Loginform();

$model->secnario = ‘login’;

}

}

?>

**ActiveForm**表单

在开发中通常需要根据字段所属数据库表对表单数据进行分组，以便分组接收数据（组名与数据表名的对应关系不是必然的）。

exp1\_1:view

<from action=’/index.php?r=course120/create’ method=’post’>

<input type=’text’ name=’user[username]’ />

<input type=’text’ name=’user[province]’ />

<input type=’text’ name=’team[name]’ />

</from>

Exp1-2:controller

Namespace app\controllers;

Use yii\web\Controller;

Use app\models\User;

Ues yii;

Class Course120Controller extens Controller{

Public $enableCsrfvalidation = false; //禁用表单验证随机码（系统生成 ）

Public function actionIndex(){

$model = new User();

If(Yii::$app -> request -> isPost){

$model -> load(Yii::$app->request->post()); //可以用批量赋值

$model -> create\_time = time(); //填充当前时间

If($model -> save() == fase){

Var\_dump($model -> getErrors());

}

}

Return $this->render(‘view’,[‘model’=>$model]);

}

}

注：如果使用load，view中的组名若与数据表名相同 ，load会将分组中的字段与数据表中相应的字段进行匹配。

在exp1\_1:view中不能使用表单验证随机码，为使用ActiveForm的随机码进行表单安全验证，改进如下：

exp1\_1\_1:view

<?php

Use yii\widgets\ActiveForm;

?>

<?php $form = ActiveForm::begin();?>

<?php ActiveForm::end();?>

一个带验证规则的form生成了如下随机验证码：



ActiveForm与ActiveRecord的配合使用

Exp1\_1:controller

<?php

Namespace app\controllers;

Use yii\web\Controller;

Use yii;

Use app\models\User;

Class Course121Controller extends Controller{

$model = new User();

If(Yii::$app->request->isPost){

$model -> load(Yii::$app -> request -> post());

$model -> create\_time = time();

If($model -> save(){

Die(‘ok’);

}

}

Return $this->render(‘create’,[‘model’=>$model]);

}

?>

Exp1\_2:view

<?php

Use yii\helpers\Html;

Use yii\widgets\ActiveForm;

?>

<div style=”margin:100px auto;width:500px;”>

<?php $form = ActiveForm::begin() ?>

<?= $form->field($model,attribute:’username’) ?>

<?= $form->field($model,attribute:’province’) ?>

<?= $form->fie($model,attribule:’team\_id) -> dropDownList([1=>’A组’, 2=>’B组’]) ?> <!--下拉框 -->

<div>

<?= Html::submitButton() ?>

</div>

<?php ActionForm::end() ?>

</div>

注：1、ActiveForm自动将表单项的name属性分组处理。

2、如果使用ActiveFrom不建议在form标签直接使用html代码，这样会造成一些后期问题。

3、表单中输入框默认是textInput();

4、ActiveForm默认开启客户端认证:

<?php $form = ActiveForm::begin([

‘enableClientScript’ => true, //加载验证js

‘enableClientValidation’=>true //开启验证

]) ?>

若需要关闭，将其值设为false;

5、关于from标签自身的属性，统一在ActiveForm::begin([])里面写，如：

<?php $from = ActiveForm::begin([

‘options’ => [

‘class’ => ‘a’,

‘id’ => ‘id’

]

]) ?>

6、关于子项的标签属性都在相应方法中写如：textInput();

7、label(label:’用户名1’)方法能够改变项目名称：如：

<?= $form->field($model,attribute:’username’) -> textInput() -> label(lebel:’用户名1’) ?>

将‘用户名’变为‘用户名1’。

8、通过hint(content:‘用户名不能为空’)，添加表单项注释。

Asset资源

资源包指继承yii\webAssetBundle的PHP类，包名为可自动加载的PHP类名， 在资源包类中，要指定资源所在位置， 包含哪些CSS和JavaScript文件以及和其他包的依赖关系。

资源根据它们的位置可分为：

* 源资源: 资源文件和PHP源代码放在一起，不能被Web直接访问，为了使用这些源资源， 它们要拷贝到一个可Web访问的Web目录中 成为发布的资源，这个过程称为*发布资源*，随后会详细介绍。
* 发布资源: 资源文件放在可通过Web直接访问的Web目录中；
* 外部资源: 资源文件放在与你的Web应用不同 的Web服务器上；

配置

通常不需要对资源配置，但资源提供了一些配置项：

Exp1:

$config = [

‘components’ =>[

‘assetManager’ => [

‘appendTimestamp’ => true //为资源添加版本号

//系统中的资源文件可以通过配置进行替换，如：

‘bundles ‘ => [

‘yii\web\JqueryAsset’ => [

‘js’ => [

//替换系统默认资源文件

‘http://baidu.com/jquery/1.8.3.js’

];

]

],

//符号连接默认关闭，不支持windows系统。启用后，当下一次资源文件有变化的时候不需要再发布，而是直接改变web下的资源

‘linkAssets’ => true,

]

]

]

参数说明

namespace app\assets;

use yii\web\AssetBundle;

Class AppAsset extends AssetBundle

{

//资源文件所在的路径

Public $basePath= ‘@webroot’; //@webroot表示web目录的物理路径

Public $baseUrl= ‘@web’; //url指向的根目录

Public $css = [

‘css/site.ss’,

];

Public $js = [

],

Public $depends = [

‘yii\web\YiiAsset’,

‘yii\bootstrap\BootstrapAsset’

];

Public $jsOptions = [

‘position’ => View::POS\_HEAD; //资源放置在页面的位置

]

}

}

使用资源

在布局文件或者视图文件引入并注册。

Exp1:

<?php

Use yii\helpers\Html;

Use app\assets\AppAsset;

AppAsset::register($this); //用资源注册当前对象

?>

<?php $this->beginPage() ?>

<!DOCTYPE html>

<html lang=”<?= Yii::$app->charset ?> “>

<?= Html::csrfMetaTags() ?>

<title><?= Html::encode($this->title) ?></title>

创建外面可访问的asset

Exp1:新建一个外面可以访问的asset

于系统根目录\assets\新建WxAsset.php

<?php

Namespace app\assets;

Use yii\web\AssetBundle;

Class WxAsset extends AssetBundle{

public $basePath = ‘@webroot’;

public $baseUrl = ‘@web’; //http://xxx.com/

public $js = [

‘js/a.js’,

‘http://baidu.com/a.js’ //指定可以访问的url，引用外部资源

];

Public $css = [];

}

注：1、建立一个外部可以访问的资源，需要必须的属性$basePath,$baseUrl和可访问的资源（$js/$css）

2、被依赖的资源会在当前资源加载之前加载。

创建一个可以发布的asset

Exp2:创建一个可以发布的asset

于系统根目录\assets\新建AAsset.php

<?php

Namespace app\assets;

Use yii\web\AssetBundle;

Class AAsset extends AssetBundle{

public $sourcePach = “@app/res/test”; //资源文件所在的路径

Public $js = []

Public $css = [];

Public $publshOptions = [ //发布控制

‘only’ => [

‘css/\*’, //\*表示所有文件

‘fonts/\*’,

‘test.js’

];

’except’ => [

‘img’ //屏蔽的资源目录

];

]

}

注：1、一个发布的资源需要指定$ourcePath 和$publshOptions属性

2、如果在only项中没有指定发布某一类文件，在except项中没有指定该类文件的目录。那么该类目录会被发布。

小技巧：配置错误提示信息为中文

$config = [

‘language’ => “zh-CN”

]

## Bower 和 NPM 资源

大多数 JavaScript/CSS 包通过[Bower](http://bower.io/) 和/或 [NPM](https://www.npmjs.org/)管理， 如果你的应用或扩展使用这些包，推荐你遵循以下步骤来管理库中的资源：

1. 修改应用或扩展的 composer.json 文件将包列入require 中， 应使用bower-asset/PackageName (Bower包) 或 npm-asset/PackageName (NPM包)来对应库。
2. 创建一个资源包类并将你的应用或扩展要使用的JavaScript/CSS 文件列入到类中， 应设置 [sourcePath](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$sourcePath-detail) 属性为@bower/PackageName 或 @npm/PackageName， 因为根据别名Composer会安装Bower或NPM包到对应的目录下。

**注意：**一些包会将它们分布式文件放到一个子目录中，对于这种情况，应指定子目录作为 [sourcePath](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$sourcePath-detail)属性值， 例如，[yii\web\JqueryAsset](http://www.yiichina.com/doc/api/2.0/yii-web-jqueryasset)使用 @bower/jquery/dist 而不是 @bower/jquery。

常用资源包

Yii框架定义许多资源包，如下资源包是最常用， 可在你的应用或扩展代码中引用它们。

* [yii\web\YiiAsset](http://www.yiichina.com/doc/api/2.0/yii-web-yiiasset): 主要包含yii.js 文件，该文件完成模块JavaScript代码组织功能， 也为 data-method 和 data-confirm属性提供特别支持和其他有用的功能。
* [yii\web\JqueryAsset](http://www.yiichina.com/doc/api/2.0/yii-web-jqueryasset): 包含从jQuery bower 包的jquery.js文件。
* yii\bootstrap\BootstrapAsset: 包含从Twitter Bootstrap 框架的CSS文件。
* yii\bootstrap\BootstrapPluginAsset: 包含从Twitter Bootstrap 框架的JavaScript 文件 来支持Bootstrap JavaScript插件。
* yii\jui\JuiAsset: 包含从jQuery UI库的CSS 和 JavaScript 文件。

如果你的代码需要jQuery, jQuery UI 或 Bootstrap，应尽量使用这些预定义资源包而非自己创建， 如果这些包的默认配置不能满足你的需求，可以自定义配置， 详情参考[自定义资源包](http://www.yiichina.com/doc/guide/2.0/structure-assets#customizing-asset-bundles)。

资源转换

除了直接编写CSS 和/或 JavaScript代码，开发人员经常使用扩展语法来编写，再使用特殊的工具将它们转换成CSS/Javascript。 例如，对于CSS代码可使用[LESS](http://lesscss.org/) 或 [SCSS](http://sass-lang.com/)， 对于JavaScript 可使用 [TypeScript](http://www.typescriptlang.org/)。

可将使用扩展语法的资源文件列到资源包的[css](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$css-detail) 和 [js](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$js-detail)中，如下所示：

**class** **AppAsset** **extends** **AssetBundle**

{

**public** $basePath = '@webroot';

**public** $baseUrl = '@web';

**public** $css = [

'css/site.less',

];

**public** $js = [

'js/site.ts',

];

**public** $depends = [

'yii\web\YiiAsset',

'yii\bootstrap\BootstrapAsset',

];

}

当在视图中注册一个这样的资源包，[asset manager](http://www.yiichina.com/doc/api/2.0/yii-web-assetmanager)资源管理器会自动运行预处理工具将使用扩展语法 的资源转换成CSS/JavaScript，当视图最终渲染页面时， 在页面中包含的是CSS/Javascipt文件， 而不是原始的扩展语法代码文件。

Yii使用文件扩展名来表示资源使用哪种扩展语法， 默认可以识别如下语法和文件扩展名：

* [LESS](http://lesscss.org/): .less
* [SCSS](http://sass-lang.com/): .scss
* [Stylus](http://learnboost.github.io/stylus/): .styl
* [CoffeeScript](http://coffeescript.org/): .coffee
* [TypeScript](http://www.typescriptlang.org/): .ts

Yii依靠安装的预处理工具来转换资源，例如， 为使用[LESS](http://lesscss.org/)，应安装lessc 预处理命令。

可配置yii\web\AssetManager::converter自定义预处理命令和支持的扩展语法， 如下所示：

**return** [

'components' => [

'assetManager' => [

'converter' => [

'class' => 'yii\web\AssetConverter',

'commands' => [

'less' => ['css', 'lessc {from} {to} --no-color'],

'ts' => ['js', 'tsc --out {to} {from}'],

],

],

],

],

];

如上所示，通过[yii\web\AssetConverter::$commands](http://www.yiichina.com/doc/api/2.0/yii-web-assetconverter#$commands-detail) 属性指定支持的扩展语法， 数组的键为文件扩展名（前面不要.）， 数组的值为目标资源文件扩展名和执行资源转换的命令， 命令中的标记 {from} 和{to}会分别被源资源文件路径和目标资源文件路径替代。

**信息：**除了以上方式，也有其他的方式来处理扩展语法资源， 例如，可使用编译工具如[grunt](http://gruntjs.com/) 来监控并自动转换扩展语法资源，此时， 应使用资源包中编译后的CSS/Javascript文件而不是原始文件。

合并和压缩资源

一个Web页面可以包含很多CSS 和/或 JavaScript 文件，为减少HTTP 请求和这些下载文件的大小， 通常的方式是在页面中合并并压缩多个CSS/JavaScript 文件为一个或很少的几个文件， 并使用压缩后的文件而不是原始文件。

**信息：**合并和压缩资源通常在应用在产品上线模式， 在开发模式下使用原始的CSS/JavaScript更方便调试。

接下来介绍一种合并和压缩资源文件 而不需要修改已有代码的方式：

1. 找出应用中所有你想要合并和压缩的资源包，
2. 将这些包分成一个或几个组，注意每个包只能属于其中一个组，
3. 合并/压缩每个组里CSS文件到一个文件，同样方式处理JavaScript文件，
4. 为每个组定义新的资源包：
   * 设置[css](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$css-detail) 和 [js](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$js-detail) 属性分别为压缩后的CSS和JavaScript文件；
   * 自定义设置每个组内的资源包，设置资源包的[css](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$css-detail) 和 [js](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$js-detail) 属性为空, 并设置它们的 [depends](http://www.yiichina.com/doc/api/2.0/yii-web-assetbundle#$depends-detail) 属性为每个组新创建的资源包。

使用这种方式，当在视图中注册资源包时，会自动触发原始包所属的组资源包的注册， 然后，页面就会包含以合并/压缩的资源文件， 而不是原始文件。

假定你的应用有两个页面X 和 Y，页面X使用资源包A，B和C，页面Y使用资源包B，C和D。

有两种方式划分这些资源包，一种使用一个组包含所有资源包， 另一种是将（A,B,C）放在组X，（B，C，D）放在组Y， 哪种方式更好？第一种方式优点是两个页面使用相同的已合并CSS和JavaScript文件使HTTP缓存更高效， 另一方面，由于单个组包含所有文件， 已合并的CSS和Javascipt文件会更大，因此会增加文件传输时间，在这个示例中， 我们使用第一种方式，也就是用一个组包含所有包。

**信息：**将资源包分组并不是无价值的，通常要求分析现实中不同页面各种资源的数据量， 开始时为简便使用一个组。

在所有包中使用工具(例如 [Closure Compiler](https://developers.google.com/closure/compiler/), [YUI Compressor](https://github.com/yui/yuicompressor/)) 来合并和压缩CSS和JavaScript文件， 注意合并后的文件满足包间的先后依赖关系， 例如，如果包A依赖B，B依赖C和D，那么资源文件列表以C和D开始， 然后为B最后为A。

合并和压缩之后，会得到一个CSS文件和一个JavaScript文件， 假定它们的名称为all-xyz.css 和 all-xyz.js，xyz 为使文件名唯一以避免HTTP缓存问题的时间戳或哈希值。

现在到最后一步了，在应用配置中配置[asset manager](http://www.yiichina.com/doc/api/2.0/yii-web-assetmanager) 资源管理器如下所示：

**return** [

'components' => [

'assetManager' => [

'bundles' => [

'all' => [

'class' => 'yii\web\AssetBundle',

'basePath' => '@webroot/assets',

'baseUrl' => '@web/assets',

'css' => ['all-xyz.css'],

'js' => ['all-xyz.js'],

],

'A' => ['css' => [], 'js' => [], 'depends' => ['all']],

'B' => ['css' => [], 'js' => [], 'depends' => ['all']],

'C' => ['css' => [], 'js' => [], 'depends' => ['all']],

'D' => ['css' => [], 'js' => [], 'depends' => ['all']],

],

],

],

];

如[自定义资源包](http://www.yiichina.com/doc/guide/2.0/structure-assets#customizing-asset-bundles) 小节中所述，如上配置改变每个包的默认行为， 特别是包A、B、C和D不再包含任何资源文件， 都依赖包含合并后的all-xyz.css 和 all-xyz.js文件的包all， 因此，对于页面X会包含这两个合并后的文件而不是包A、B、C的原始文件， 对于页面Y也是如此。

最后有个方法更好地处理上述方式，除了直接修改应用配置文件， 可将自定义包数组放到一个文件，在应用配置中根据条件包含该文件， 例如：

**return** [

'components' => [

'assetManager' => [

'bundles' => **require**(**\_\_DIR\_\_** . '/' . (YII\_ENV\_PROD ? 'assets-prod.php' : 'assets-dev.php')),

],

],

];

如上所示，在产品上线模式下资源包数组存储在assets-prod.php文件中， 不是产品上线模式存储在assets-dev.php文件中。

笔记完于2017年10月3日星期二

笔记修改于2017年11月13日星期一