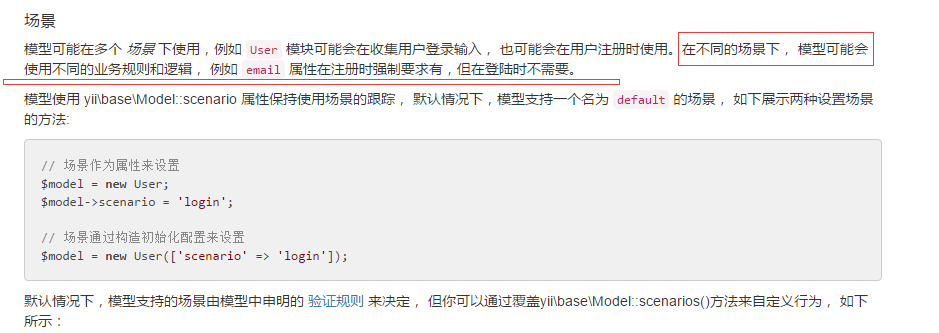
## 应用结构

1.(应用结构/控制器) 控制器的注意

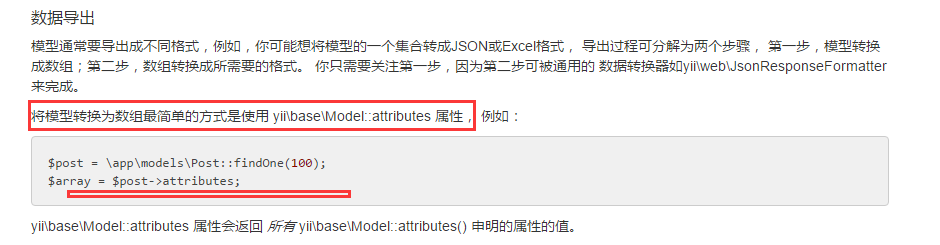


1. (应用结构/模型) 对于模型的场景要求

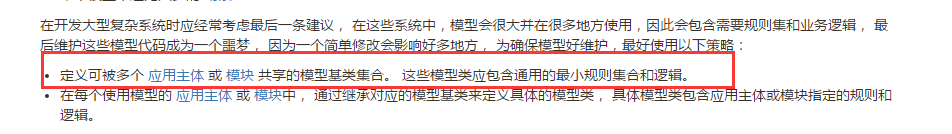
Scenario 场景理解



对于对象属性转换成方法，可以使用基类中的attributes方法



对于模型抽出基类



1. 对于模块(应用结构/模块)

Module 继承 \yii\base\Module 继承 \yii\di\ServiceLocator\ 继承 \yii\base\Component

继承 \yii\base\Object

\yii\base\Module中对于一个模块基础功能属性的设定

对于controller中的方法是如何使用模块?

1. 过滤器

可以在一个控制器中进行设定

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');

},

],

];

}

对于跨域的配置实例

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());

}

1. 扩展 composer使用

运行Composer.phar ，读取 composer.json的数据，寻找最新的版本包，最后生成 composer.lock文件，

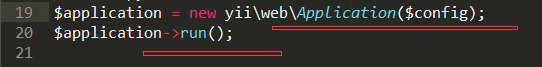
Composer.lock文件记录了最终的安装信息，可以与其他人共享composer.lock,这样就会配置相同的环境，

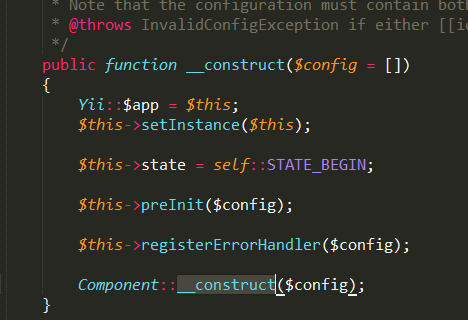
如果之后想更新的话，就重新使用composer.json里的元数据，重新生成composer.lock包。

## 请求处理

1. 引导







1. 路由信息
2. 缺省路由 -->默认路由

**return** [ // ... 'defaultRoute' => 'main/index', ];

1. 拦截路由 --> 例如当网站进行维护时，进行拦截跳转

**return** [ // ... 'catchAll' => ['site/offline'], ];

1. 路由美化

[

'components' => [

'urlManager' => [

'enablePrettyUrl' => **true**,//开启路由美化

'showScriptName' => **false**,//不显示index.php脚本名称

'enableStrictParsing' => **false**,

'rules' => [

// ...

],

],

],

]

1. 请求(requests)

你可以调用 request 组件的 yii\web\Request::get() 方法和 yii\web\Request::post() 方法。 他们分别返回 $\_GET 和 $\_POST 的值。

可以通过 yii\web\Request::headers 属性返回的 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 \*/ }

1. 错误处理(这里可以理解:对于一个控制器，可以使用actions方法进行一些信息的跳转，例如错误信息，可以定义‘error’ =>['class' => 'yii\web\ErrorAction'], )如下:

**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 类， 该类渲染名为error视图来显示错误。

1. 日志处理

yii\log\Dispatcher::flushInterval 和 yii\log\Target::exportInterval 都为1， 就像下面这样：

**return** [

'bootstrap' => ['log'],

'components' => [

'log' => [

'flushInterval' => 1, //一般情况，日志都是累计一定量才会输出，使用该参数可以每1条日志输出一次。默认1000

'traceLevel' => YII\_DEBUG ? 3 : 0,//当为调试模式时，向上最多追加3个堆栈

'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