Update

*从yii2.0.14版本开始记录*

* v2.0
* v2.0.14

http:*//nai8.me/forum/detail.html?id=42*

* v2.0.14.2

http:*//nai8.me/article/detail.html?id=345*

Composer

* 基本用法
* 安装yii程序

composer **create**-**project** *--prefer-dist yiisoft/yii2-app-basic basic*

* 通过composer.json安装扩展

composer **install**

* 更新本地composer扩展库

composer **update**

* 直接安装某个composer扩展

**composer** **require** [options] [--] [vendor/packages]...

DAO

*Yii的数据库读取对象，在PDO之上，DAO后有了Query Builder和AR*

* 基本使用方法
* 获得数据库连接

**$**conn = Yii::$app->db;

* 执行数据库查询语句
* Yii::$app->db->createCommand("SELECT \* FROM `user`");
* Yii::$app->db->createCommand("SELECT \* FROM `user` WHERE uid=:uid",[":uid"=>1]);

Yii::$app->db->createCommand("SELECT \* FROM `user` WHERE uid=:uid")->addValue([":uid"=>1]);

* SQL语句插入数据
* Yii::$app->db

->createCommand('INSERT INTO user (email, password) VALUES("test3@example.com", "test3");')->execute();

* 数组形式插入数据
* Yii::$app->db->createCommand()->insert('user', [
* 'email' => 'test4@example.com',
* 'password' => 'changeme7',
* 'first\_name' => 'Test'

])->execute();

* 批量插入数据
* Yii::$app->db->createCommand()->batchInsert('user', ['email', 'password', 'first\_name'],
* [
* ['james.franklin@example.com', 'changeme7', 'James'],
* ['linda.marks@example.com', 'changeme7', 'Linda']
* ['roger.martin@example.com', 'changeme7']

])->execute();

* 更新数据

Yii::$app->db->createCommand()->update('user', ['updated\_at' => time()], 'id = 2')->execute();

* 删除数据

Yii::$app->db->createCommand()->delete('user', 'id = 3')->execute();

* 获取结果方法
* 获取所有数据(数组形式返回)
* Yii::$app->db->createCommand("SELECT \* FROM `user`")->queryAll();
* 获取一条数据（一维数组）

Yii::$app->db->createCommand("SELECT \* FROM `user` WHERE id = 1")->queryOne();

* 获取一个值

Yii::$app->db->createCommand("SELECT count(\*) AS total FROM `user` WHERE id = 1")->queryScalar();

* 获取某一列（放到一位数组中）

Yii::$app->db->createCommand("SELECT username FROM `user`")->queryColumn();

Logging

*日志功能*

* 配置Log
* 基本用法
* trace

Yii::trace($message,$category)

* info

Yii::info($message,$category)

* warning

Yii::warning($message,$category)

* error

Yii::error($message,$category)

Validator

*数据验证，最常用于模型的rules()函数*

* 方法列表（Model中rules函数）
* required 必须值
* [**"username",'required'**]
* [**["username","email"**],'required']
* [**["username"**],'required',"message"=>"{attribute}必须填写"]

[**["username"**],'required','requiredValue'=>"abei"] *// 用户填写的值必须等于requiredValue才能通过验证。*

* Email验证
* ["email",'email']
* [["email","work\_email"],'email']
* Boolean验证

['sex', 'boolean', 'trueValue' => **true**, 'falseValue' => **false**, 'strict' => **true**];*// 可以认为置顶 true / false 值。*

* captcha验证码

['verificationCode', 'captcha'];

* compare比较
* ['username', 'compare', 'compareAttribute' => 'province','message'=>'username和province必须一样'] *//错误信息将提示给username*
* ['age', 'compare', 'compareValue' => 30, 'operator' => '>=','type' => 'number'];*//compareValue:比较常量值 operator:比较操作符 type为值类型，默认为string，会一个每个字符对比，若为number则直接判断数值*

*// operator 待选值==、===、!=、!==、>、>=、<、<=*

* date验证

["birth","date","format"=>"Y-m-d"]

* default验证
* ['age','default','value'=>**null**] *// 当age为空的时候设置为null*
* ['country','default','value'=>'USA'] *// 当 country为空时设置为USA*
* */\* 如果from为空，则=今天+3天，如果to为空，则=今天+6天 \*/*
* [['from','to'],'default','value'=>**function**($model,$attribute){
* **return** date('Y-m-d', strtotime($attribute === 'to' ? '+3 days' : '+6 days'));
* }]
* double/number验证
* ['v','double'] *// 判断v是否为数字*

['v','double','max'=>90,'min'=>1]*//判断v是否为数字且大于等于1、小于等于90*

* 数组各元素验证
* */\* 要求验证的元素必须为数组，否则会返回假并报错 \*/*

["categoryIds","each","rule"=>['integer']]

* exist是否存在验证
* */\* 所谓对存在的检查实质为where的与操作，必须同时瞒住的记录存在方可。兄弟们可以研究下，exist是对sql语句EXISTS的应用\*/*
* ["username","exist"] *// username输入的值已经存在*
* ["username","exist","targetAttribute"=>"province"] *// username的输入值必须在province列存在*
* ["username","exist",'targetAttribute' => ['username', 'province']] *// username的输入值必须在username和province中存在*

[["username","province"],"exist",'targetAttribute' => ['username', 'province']] *// username和province的输入值必须在username和province中存在*

* file验证
* */\* maxFiles代表一次最多传几个，mimeTypes代表上传文件类型 \*/*

['primaryImage', 'file', 'extensions' => ['png', 'jpg', 'gif'],'mimeTypes'=>["image/\*"], 'maxSize' => 1024\*1024,'minSize'=>100\*1024,'maxFiles'=>6,'checkExtensionByMimeType'=>**true**],

* filter过滤验证函数
* [['username', 'email'], 'filter', 'filter' => 'trim', 'skipOnArray' => **true**],
* ['phone', 'filter', 'filter' => **function** ($value) {
* *// normalize phone input here*
* **return** $value;

}],

* image验证
* */\* 上传png/jpg格式，最大宽度不能超过1000px，最小宽度不能低于100px，最大高度不能高于1000px。最小高度不能低于100px \*/*

['primaryImage', 'image', 'extensions' => 'png, jpg','minWidth' => 100, 'maxWidth' => 1000,'minHeight' => 100, 'maxHeight' => 1000]

* ip验证
* ["ip\_addess","ip"]
* in方法验证

["level","in","range"=>[1,2,3]]

* integer验证
* ["age",'integer'];

["age","integer","max"=>90,"min"=>1]

* 正则匹配验证

["username","match","pattern"=>"/^[a-z]\w\*$/i"]

* safe验证（多用于设置一个model的attribute）

["description","safe"]

* string验证
* [**"username","string","length"=>[4,24**]];
* [**"username","string","min"=>4**];
* [**"username","string","max"=>32**];

[**"username","string","encoding"=>"UTF-8"**];

* unique唯一验证
* ["username","unique"]

["username","unique","targetAttribute"=>"province"]

* url验证
* ["website","url"]

["website","url","validSchemes"=>["http","https"]]

String

*字符串*

* 基础用法
* 一个字符串中单词数量

StringHelper::countWords("hello world");//2

* 返回路径中的文件名部分

StringHelper::basename("/path/hello.txt",".txt"); // hello

* 返回路径中的目录名

StringHelper::dirname("/home/path/hello.txt");// /home/path

* 超出内容用...代替(不含HTML)

StringHelper::truncate("hello world",7,'...'); //hello w...

* 超出内容用...代替(识别HTML)

StringHelper::**truncate**("**hello world**",7,'...',null,true);//**hello w**...

* 以单词为单位超出部分用..代替（不解析HTML）

StringHelper::truncateWords('This is a test sentance', 4, '...') *//This is a test ...*

* 以单词为单位超出部分用..代替（解析HTML）

StringHelper::truncateWords('This is a test **for a sentance**', 5, '...',**true**) *//This is a test* ***for****...*

* 一个字符串是否以另一个字符串开始

StringHelper::startsWith("hello world","he");//true

* 一个字符串是否以另一个字符串结尾

StringHelper::endsWith("hello world","ald");//false

* 按照分隔符分隔字符串为数组
* StringHelper::explode('It, is, a first, test'));//['It','is','a first','test']
* StringHelper::explode("a@b@c","@");['a','b','c']

StringHelper::explode("a, b ,c ");['a','b','c']

Session&Cookie

*Session被封装成一个应用组件，直接通过Yii::$app->session来访问；Cookie通过Request和Response来操作。*

* Session
* 获得session

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

* 检查session是否开启

Yii::$app->session->isActive

* 开启一个session

Yii::$app->session->open()

* 关闭session

Yii::$app->session->close();

* 销毁session中所有已注册的数据

Yii::$app->session->destroy();

* 访问一个session
* */\* 以下三种方法效果等同 \*/*
* $language = $session->get('language');
* $language = $session['language'];

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

* 设置一个session
* /\* 以下三种方法效果等同 \*/
* **$**session->set('language', 'en-US');
* **$**session['language'] = 'en-US';

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

* 删除一个session变量
* */\* 下面三种方法效果等同 \*/*
* $session->remove('language');
* **unset**($session['language']);

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

* 检查一个session变量是否存在
* */\* 以下三种方法效果一致 \*/*
* **if** ($session->has('language')) ...
* **if** (**isset**($session['language'])) ...

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

* Cookie
* 获取cookie

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

* 设置cookie

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

* 获取一个cookie值

**$**language = $cookies->getValue('language', 'en');// 如果获取language失败，则返回"en"代替

* 另一种获取cookie值方法
* **if** (($cookie = $cookies->get('language')) !== **null**) {
* $language = $cookie->value;

}

* 数组方式获取cookie值
* **if** (**isset**($cookies['language'])) {
* $language = $cookies['language']->value;
* }
* 检查一个cookie是否存在
* **if** ($cookies->has('language')) ...

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

* 新增一个cookie
* $cookies->add(**new** \yii\web\Cookie([
* 'name' => 'language',
* 'value' => 'zh-CN',

]));

* 删除一个cookie
* **$**cookies->remove('language');

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

Request

*Request 被配置为一个应用组件，我们可以通过Yii::$app->request访问它。*

* URL相关
* 获得当前请求的绝对url

Yii::$app->request->getAbsoluteUrl();

* 返回一个请求URL的hostInfo部分

Yii::$app->request->getHostInfo();

* 获得URL问号后的参数字符串

Yii::$app->request->getQueryString()

* 返回服务器端口

Yii::$app->request->getServerPort();

* HTTP头
* 返回用户接受的内容类型

Yii::$app->request-> getAcceptableContentTypes ();// Header Accept

* 返回用户可接受的语言

Yii::$app->request-> getAcceptableLanguages(); // Header Accept-Language

* 返回GET/POST请求
* Yii::$app->request->get();
* Yii::$app->request->get("id");
* Yii::$app->request->POST();

Yii::$app->request->POST("username");

* 判断请求类型（返回boolean）
* Yii::$app->request->isAjax *// 判断是否为ajax请求*
* Yii::$app->request->isConsoleRequest *// 判断是否为控制发起的请求*
* Yii::$app->request->isDelete *// 判断是否为DELETE请求*
* Yii::$app->request->isGet *// 判断是否为GET请求*
* Yii::$app->request->isPost *// 判断是否为POST请求*

Yii::$app->request->isPjax *// 判断是否为isPjax请求*

* 客户端信息
* 返回用户的 IP

Yii::$app->request->getUserIP();

Response

*和Request一样，Response被封装成Yii的一个组件，你可以通过Yii::$app->response轻松的访问它。*

* Status Code状态码
* 设置一个Status Code

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

* Yii内置的通过异常形式返回状态码
* yii\web\BadRequestHttpException: status code 400.
* yii\web\ConflictHttpException: status code 409.
* yii\web\ForbiddenHttpException: status code 403.
* yii\web\GoneHttpException: status code 410.
* yii\web\MethodNotAllowedHttpException: status code 405.
* yii\web\NotAcceptableHttpException: status code 406.
* yii\web\NotFoundHttpException: status code 404.
* yii\web\ServerErrorHttpException: status code 500.
* yii\web\TooManyRequestsHttpException: status code 429.
* yii\web\UnauthorizedHttpException: status code 401.

yii\web\UnsupportedMediaTypeHttpException: status code 415.

* 抛出其他Status Code
* **throw** **new** \yii\web\HttpException(402); *// 如果系统没有，可以通过HttpException自己写状态码*

**throw** **new** \yii\web\HttpException(402,"message");

* HTTP Headers
* 添加设置删除Http Headers内容
* $headers = Yii::$app->response->headers;
* *// add a Pragma header. Existing Pragma headers will NOT be overwritten.*
* $headers->add('Pragma', 'no-cache');
* *// set a Pragma header. Any existing Pragma headers will be discarded.*
* $headers->set('Pragma', 'no-cache');
* *// remove Pragma header(s) and return the removed Pragma header values in an array*

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

* Response Body
* 相应主体

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

Controller

*控制器，可在action内直接用$this调用。*

* 视图相关
* 渲染一个视图（如果布局有效则使用布局）

**$this**->render('index',['model'=>$model])

* 渲染视图（不使用布局）

**$this**->renderPartial('index',['model'=>$model])

* 渲染视图（不使用布局）
* *// 注入所有注册的JS/CSS脚本和文件,通常使用在响应AJAX网页请求的情况下*

**$this**->renderAjax('index',['model'=>$model])

* 只渲染布局

**$**this->renderContent($content);

ActiveForm

*重点！列出最常用的ActiveForm方法。*

* 常用方法
* 取消客户端规则验证
* $form = ActiveForm::**begin**([
* 'enableClientValidation'=>false

]);

* 取消yii.js的引入
* $form = ActiveForm::**begin**([
* 'enableClientScript'=>false

]);

* 表单目标地址设置
* $form = ActiveForm::**begin**([
* "action"=>$url

])

* GET & POST 方法设置
* $form = ActiveForm::**begin**([
* "method"=>"POST"

]);

* 设置Form的类及自己定义标签属性
* $form = ActiveForm::begin([
* 'options'=>["class"=>"f","data-name"=>"xxx"]

]);

* 生成文本框

$form->field($model, 'date')->textInput(["key"=>"value"]);

* 生成文本域

$form->field($model, 'date')->textarea(["key"=>"value"]);

* 单选列表

$form->field($model,'sex')->radioList($arr,["key"=>"value"]);

* 密码框

$form->field($model,"password")->passwordInput();

* 复选框

$form->field($model,"city\_id")->checkboxList($arr);

* 文件上传

$form->field($model,"image")->fileInput();

* 隐藏域

$form->field($model,"name")->hiddenInput();

FileHelper

*几个常用也好用的文件帮助方法*

* 基本方法
* 遍历一个文件夹下文件&子文件夹
* FileHelper::findFiles('/path/to/search/');
* FileHelper::findFiles('.', ['only' => ['\*.php', '\*.txt']]); *// 只返回php和txt文件*
* FileHelper::findFiles('.', ['except' => ['\*.php', '\*.txt']]); *// 排除php和txt文件*
* 获得指定文件的MIME类型

FileHelper::getMimeType('/path/to/img.jpeg');

* 复制文件夹

FileHelper::copyDirectory($src, $dst, $options = [])

* 删除一个目录及内容

FileHelper::removeDirectory($dir, $options = [])

* 生成一个文件夹（同时设置权限）

FileHelper::createDirectory($path, $mode = 0775, $recursive = true)

UploadedFile

*上传文件帮助类*

* 基本函数
* 通过模型的属性获取一个文件

**$**file = UploadedFile::getInstance($model,'avatar')

* 通过模型的属性来获取一组文件
* $files = UploadedFile::getInstances($model,'avatar')

//view $form->field($model,'avatar[]')->fileInput()

* 通过名字上传一个文件

**$**file = UploadedFile::getInstanceByName('avatar');

* 通过名字获取一组上传的文件
* $file = UploadedFile::getInstancesByName('avatar');

// view Html::fileInput('avatar[]')

* 保存一个文件

$file->saveAs(Yii::getAlias("@webroot").'/data/test.jpg');

* 获取上传文件原始名(不含扩展名)

**$**file->getBaseName();//test.jpg ===> test

* 获取上传文件的扩展名(已经自动格式化为小写)

$file->getExtension();*// 是png、不是image/png*

* 变量说明
* 获取文件的原始名

$file->name;*//test.jpg*

* 获取文件媒体类型

**$**file->type;// image/png

* 获取文件临时名

**$**file->tempName;

* 获取文件大小

$file->size;*// 21744*

Html

*通过Html类的一些静态方法生成Html标签。*

* 生成Html标签方法
* 生成一个超级链接

Html::a('链接的文本', $url);

* 通过Yii2的路由生成一个链接
* Html::a('链接文本', Url::to(['/site/index'], **true**));

Html::a('链接文本', Yii::$app->urlManager->createUrl(['/site/index']));

* 生成一个图片链接

Html::img("/images/logo.png",['class'=>'img']);

* 生成一个按钮

Html::button("按钮文本",['class'=>'button-action']);

* 发送邮件链接

Html::mailto("阿北",'abei@nai8.me',$options);

* 生成有序列表
* $list = ['china','usa'];

Html::ol($list);

* 生成无需列表
* $list = ['china','usa','japan'];

Html::ul($list);

* 生成javascript代码

Html::script("alert('hello world');")

* 生成style代码
* **Html**::style("color:**#F60**");

**Html**::style(".list {background:#FFF;}");

* 文件引用及编码
* 生成一个css引用链接

Html::cssFile("http://baidu.com/style.css",[]);

* 生成一个js文件引用

Html::jsFile($url,[]);

* 把字符 "<" （小于）和 ">" （大于）转换为HTML实体

Html::encode($html);

* 将特色的HTML实体转化为>和<

Html::decode($string);

Alias

* 定义和使用
* 定义一个别名

**Yii**::setAlias('@**baidu**', 'http://www.baidu.com');

* 获得一个别名

Yii::getAlias($name);

* 获得Yii框架所在的目录

Yii::getAlias('@yii')

* 正在运行的应用的根目录

Yii::getAlias('@app')

* Composer第三方库所在目录

Yii::getAlias("@vendor")

* bower库所在位置

**Yii**::getAlias("@**bower**");

* npm库所在位置

**Yii**::getAlias("@**npm**");

* 运行时存放文件路径

**Yii**::getAlias("@**runtime**");

* index.php所在目录

**Yii**::getAlias("@**webroot**");

* 当前应用的根URL，主要用于前端。

**Yii**::getAlias("@**web**");

* 高级版-通用文件夹

**Yii**::getAlias("@**common**");

* 高级版-前台应用所在位置

**Yii**::getAlias("@**frontend**");

* 高级版-后台应用所在位置

**Yii**::getAlias("@**backend**");

* 命令行库所在位置

**Yii**::getAlias("@**console**");

Query Builder

*主要解决DAO在查询语句上的繁琐问题，无需输入原生SQL语句就可以完成数据库检索。*

* 基本用法
* 使用Query Builder需要使用的类

$query = (**new** \yii\db\Query()); *// yii2使用Query对象来采集SQL的各个部分，然后由Query Builder组成SQL语句后由DAO发给数据库获得请求。*

* SELECT方法
* $query->select("id,username");*// 字符串形式*
* $query->select(['id','username']); *// 数组形式*
* $query->select(["userId"=>"id","fName"=>"user.frist\_name"]); *// 起别名*

$query->select(["full\_name"=>"CONCAT(id,'-',username)"]); *// 支持MYSQL函数*

* FROM方法
* $query->from("user"); *// 字符串形式*

$query->from(["u"=>"user"]); *// 数据表别名*

* 过滤掉重复记录

$query->select("username")->distinct()->from("user"); *// distinct*

* WHERE函数用法
* */\* 传递字符串 \*/*
* $query->where("id = 1");
* $query->where("id = :id")->addParams([":id"=>1]);
* $query->where("id = :id",[":id"=>1]);
* */\* 传递数组 \*/*
* $query->where(["username"=>"abei","age"=>[20,19,26]])->from("user");*// select \* from user where username="abei" AND age in (20,19,26)*
* */\* 操作符 \*/*
* $query->where([">","id",10]);*// id > 10*
* $query->where(["<","id",10]); *// id < 10*
* $query->where(["<>","id",10]); *// id <> 10*
* $query->where(["in","id",[10,12]]);*// id in (10,20)*
* $query->where(["not in","id",[10,12]]);*// id not in (10,20)*
* $query->where(["and","id=1","id=2"]); id=1 **AND** id=2
* $query->where(['or', ['type' => [7, 8, 9]], ['id' => [1, 2, 3]]]); *// (type IN (7, 8, 9) OR (id IN (1, 2, 3)))*
* $query->where(["between", 'id', 1, 10]);*// id between 1 AND 10*
* $query->where(["not",["id"=>5]]);*// not (id=5)*
* $query->where(["not between","id",1,10]);*// id not between 1 AND 10*
* $query->where(["like","username","abei"]); *// username like "%abei%"*
* $query->where([['like', 'username', ['abei', 'liuhuan']]]); *// username like "%abei%" AND username like "%liuhuan%"*
* $query->where(['like', 'username', '%abei', **false**]); *// username like "%abei"*
* $query->where(["or like", 'username', ['abei', 'liuhuan']]);*// username like "%abei%" OR username like "%liuhuan%"，只作用于范围为数组的形式*
* $query->where(["not like",xxxxx]);*// 与like用法一致*
* $query->where(["or not like",xxx])*// 与not like用法一致*
* 一个要单独说明的exists
* */\* EXISTS用于检查子查询是否至少会返回一行数据，该子查询实际上并不返回任何数据，而是返回值True或False \*/*

$query->where(['exists', (**new** Query())->select('id')->from('user')->where(['id' => 1])]);

* ORDER BY 方法
* $query->orderBy("id DESC");
* $query->orderBy(["id"=>SORT\_DESC]);

$query->orderBy(["id"=>SORT\_DESC,'create\_time'=>SORT\_ASC]);

* GROUP BY && HAVING
* $query->groupBy(["username"]);

$query->groupBy(["id"])->having([">",'id',20]);

* 获取生成的SQL语句

$query->createCommand()->sql;

* 获得查询结果
* 获取所有结果

$query->all();*// 二位数组*

* 获取一条记录

**$**query->one();

* 检查一个数据库中是否含有某个表

(**new** \yii\db\Query)->from('user')->exists();

* 获取count

**$**query->count();

* 获取一个值

**$**query->scalar();

* 获取一列值

$query->column();*// 一位数组*

* 一个例子
* 获取一个user表的内容
* **$**query = new \yii\db\Query;
* **$**query->from("user");
* **$**query->select(["fname"=>"username"]);
* **$**query->where([">",'id',10]);

**$**query->all();

Migrate

*数据库迁移工具*

* 基本方法
* 生成一个迁移文件

./yii migrate/create script\_name *// script\_name为脚本名字（需要英文格式）*

* 执行所有没有迁移的脚本
* ./yii migrate

./yii migrate/up

* 执行置顶的迁移

./yii migrate/up 脚本名 *// 不用含有扩展名*

Asset Management

*Asset资源管理*

* 常用参数
* 类属性说明
* **$**basePath // 资源文件所在的web服务器目录路径，一般为@webroot
* **$**baseUrl // js和css文件相对url基地址
* **$**css // asset bundle 所包含的css文件数组
* **$**cssOptions // 对link标签的属性控制
* **$**js // asset bundle 所包含的js文件数组
* **$**jsOptions // 对script标签的属性控制
* **$**publishOptions // 发布操作

**$**sourcePath // 当资源网络不可以访问，则必须指定此目录。

* 关键参数
* 去掉浏览器缓存

'appendTimestamp' => **true** *// 在web.php里的components - assetManager*

* 发布资源筛选
* **public** $publishOptions = [
* 'only' => [
* 'fonts/\*',
* 'css/\*',
* 'test.js'
* ],
* 'except'=>[
* 'img'
* ],

];

* js文件在页面的位置
* **public** $jsOptions = ['position' => \yii\web\View::POS\_HEAD];*//js文件发布到head标签内*
* **public** $jsOptions = ['position' => \yii\web\View::POS\_END];*//js文件发布到body标签底部*

**public** $jsOptions = ['position' => \yii\web\View::POS\_BEGIN];*//js文件放到body标签开始处*

* 浏览器兼容问题

**public** $cssOptions = ['condition' => 'IE 11'];*// 代表兼容ie11*

* 是否使用符号链接

'linkAssets' => **true** *// 在web.php里的components - assetManager*

* 配置yii自身的asset资源
* *// 在web.php里的components - assetManager，配置自定义的也可以*
* 'bundles' => [
* 'yii/web/YiiAsset'=>[
* 'js'=>[],
* ......
* ]

]

Event

*有关事件的所有，系统自带事件通通给你。*

* Application # 应用主体
* 应用处理请求before之前触发

Application::EVENT\_BEFORE\_REQUEST

* 应用处理请求before之后触发

Application::EVENT\_AFTER\_REQUEST

* Controller # 控制器
* 在每个Action运行之前触发

Controller::EVENT\_BEFORE\_ACTION

* 在每个Action运行之后触发

Controller::EVENT\_AFTER\_ACTION

* Model # 模型
* 在验证Model属性之前触发

Model::EVENT\_BEFORE\_VALIDATE

* 在验证Model属性之后触发

Model::EVENT\_AFTER\_VALIDATE

* Module # 模块
* 一个模块的Action运行前触发

Module::EVENT\_BEFORE\_ACTION

* 一个模块的Action运行后触发

Module::EVENT\_AFTER\_ACTION

* View # 视图
* 执行视图的beforePage时触发

View::EVENT\_BEGIN\_PAGE

* 执行视图的endPage函数时触发

View::EVENT\_END\_PAGE

* 在renderFile渲染一个视图文件之前触发

View::EVENT\_BEFORE\_RENDER

* 在renderFile渲染一个视图文件之后触发

View::EVENT\_AFTER\_RENDER

* 执行视图的beginBody函数时触发

View::EVENT\_BEGIN\_BODY

* 执行视图的endBody函数时触发

View::EVENT\_END\_BODY

* Widget # 挂件
* Widget初始化时触发

Widget::EVENT\_INIT

* Widget执行前触发

Widget::EVENT\_BEFORE\_RUN

* Widget执行之后触发

Widget::EVENT\_AFTER\_RUN

* ActiveQuery
* 由ActiveQuery的init函数触发

ActiveQuery::EVENT\_INIT

* BaseActiveRecord & ActiveRecord # 这也许是内置事件中最重要的一批了。
* AR对象被初始化init时触发

BaseActiveRecord::EVENT\_INIT

* AR执行查询结束时触发

BaseActiveRecord::EVENT\_AFTER\_FIND

* 插入结束时触发

BaseActiveRecord::EVENT\_BEFORE\_INSERT

* 插入之后触发

BaseActiveRecord::EVENT\_AFTER\_INSERT

* 更新记录之前触发

BaseActiveRecord::EVENT\_BEFORE\_UPDATE

* 更新记录之后触发

BaseActiveRecord::EVENT\_AFTER\_UPDATE

* 删除记录之前触发

BaseActiveRecord::EVENT\_BEFORE\_DELETE

* 删除记录之后触发

BaseActiveRecord::EVENT\_AFTER\_DELETE

* 在数据refresh成功之后触发

BaseActiveRecord::EVENT\_AFTER\_REFRESH

* Connection # 数据库连接
* 数据库连接被打开后触发

Connection::EVENT\_AFTER\_OPEN

* 事务被启动时触发

Connection::EVENT\_BEGIN\_TRANSACTION

* 事务被提交后触发

Connection::EVENT\_COMMIT\_TRANSACTION

* 事务回滚后触发

Connection::EVENT\_ROLLBACK\_TRANSACTION

* Response # Http响应
* Response响应发送之前触发

Response::EVENT\_BEFORE\_SEND

* Response响应发送之后触发

Response::EVENT\_AFTER\_SEND

* Response响应内容准备好之后触发

Response::EVENT\_AFTER\_PREPARE

* User # 会员登陆授权
* 登陆之前触发

User::EVENT\_BEFORE\_LOGIN

* 登陆之后触发

User::EVENT\_AFTER\_LOGIN

* 注销之前触发

User::EVENT\_BEFORE\_LOGOUT

* 注销之后触发

User::EVENT\_AFTER\_LOGOUT