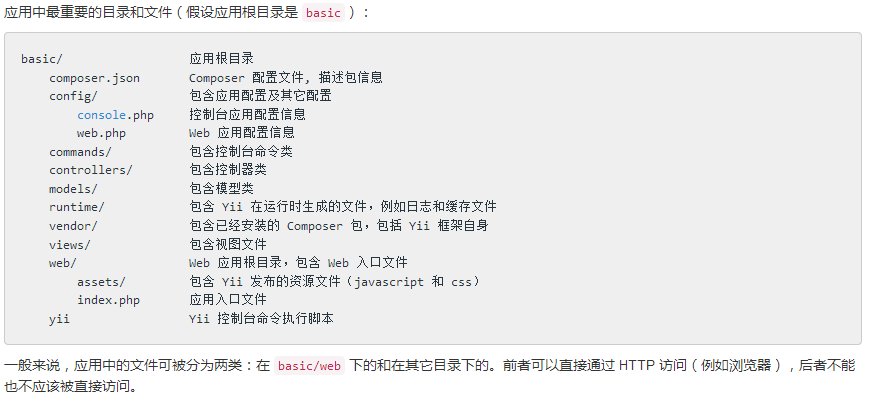
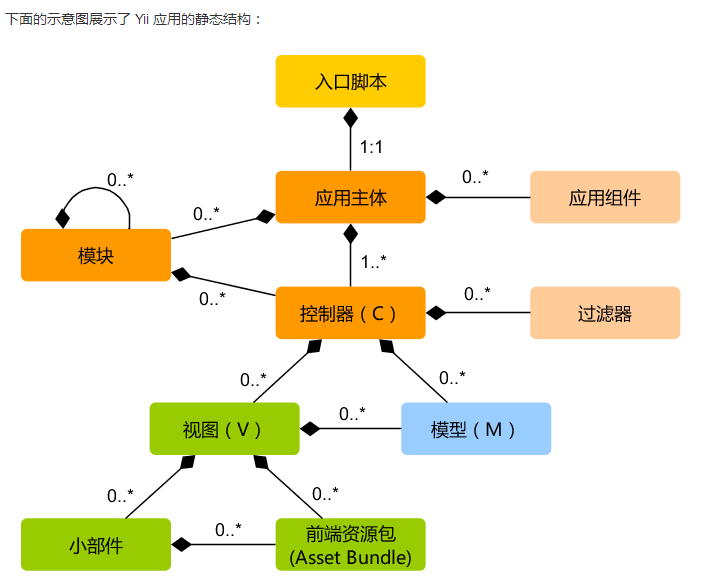
**零、Yii框架的应用结构**



上面还要有一个modules文件夹，来放模块

Yii 实现了模型-视图-控制器 (MVC)设计模式，这点在上述目录结构中也得以体现。 models 目录包含了所有模型类， views 目录包含了所有视图脚本， controllers 目录包含了所有控制器类。

模型代表数据、业务逻辑和规则； 视图展示模型的输出；控制器 接受出入并将其转换为模型和视图命令。



除了 MVC, Yii 应用还有以下部分：

入口脚本：终端用户能直接访问的 PHP 脚本， 负责启动一个请求处理周期。

应用：能全局范围内访问的对象， 管理协调组件来完成请求.

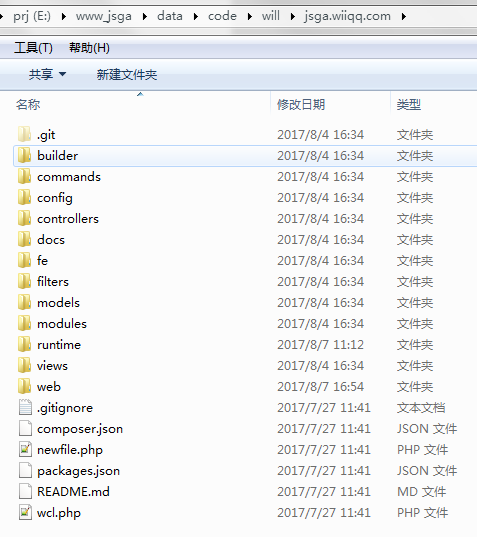
应用组件：在应用中注册的对象， 提供不同的功能来完成请求。

模块：包含完整 MVC 结构的独立包， 一个应用可以由多个模块组建。

过滤器：控制器在处理请求之前或之后 需要触发执行的代码。

小部件：可嵌入到视图中的对象， 可包含控制器逻辑，可被不同视图重复调用。

Jsga的应用结构：根目录是jsga.wiiqq.com：



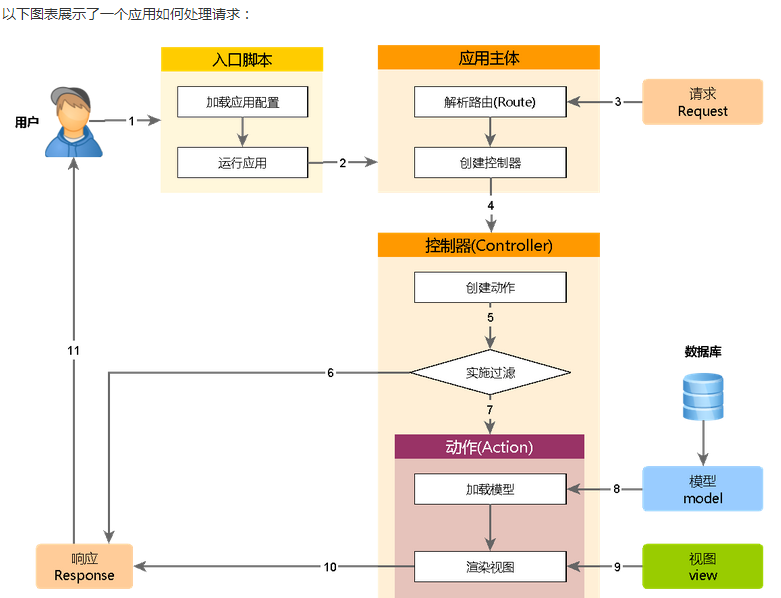
每个应用都有一个入口脚本 web/index.php， 这是整个应用中唯一可以访问的 PHP 脚本。 入口脚本接受一个 Web 请求并创建应用实例去处理它。 应用在它的组件辅助下解析请求， 并分派请求至 MVC 元素。视图使用小部件去创建复杂和动态的用户界面。

yii中命名空间和文件路径的关系:

几乎每一个核心类都引入了命名空间， 比如 yii\web\Request。当前的命名方案与目录结构相吻合。 例如，yii\web\Request 就表明对应的类文件是 Yii 框架文件夹下的 web/Request.php 文件。

有了 Yii 的类自动加载器，你可以直接使用全部核心类而不需要显式包含 具体文件。

请求生命周期：



1、用户向入口脚本 web/index.php 发起请求。

2、入口脚本加载应用配置 并创建一个应用实例去处理请求。

3、应用通过请求组件 解析请求的路由。

4、应用创建一个控制器实例去处理请求。

5、控制器创建一个动作实例并针对操作执行过滤器。

6、如果任何一个过滤器返回失败，则动作取消。

7、如果所有过滤器都通过，动作将被执行。

8、动作会加载一个数据模型，或许是来自数据库。

9、动作会渲染一个视图，把数据模型提供给它。

10、渲染结果返回给响应组件。

入口脚本

一个应用（不管是网页应用还是控制台应用）只有一个入口脚本。 终端用户的请求通过入口脚本实例化应用并将将请求转发到应用。

Web 应用的入口脚本必须放在终端用户能够访问的目录下， 通常命名为 index.php， 也可以使用 Web 服务器能定位到的其他名称。

控制台应用的入口脚本一般在应用根目录下命名为 yii（后缀为.php）， 该文件需要有执行权限，这样用户就能通过命令 ./yii <route> [arguments] [options] 来运行控制台应用。

（入口脚本有两种？）

入口脚本主要完成以下工作：

* 定义全局常量；
* 注册 [Composer 自动加载器](http://getcomposer.org/doc/01-basic-usage.md#autoloading)；
* 包含 [Yii](http://www.yiichina.com/doc/api/2.0/yii) 类文件；
* 加载应用配置；
* 创建一个[应用](http://www.yiichina.com/doc/guide/2.0/structure-applications)实例并配置;
* 调用 [yii\base\Application::run()](http://www.yiichina.com/doc/api/2.0/yii-base-application#run%28%29-detail) 来处理请求。

Web 应用

以下是基础应用模版入口脚本的代码：

<?php

defined('YII\_DEBUG') or define('YII\_DEBUG', true);

defined('YII\_ENV') or define('YII\_ENV', 'dev');

// 注册 Composer 自动加载器

require(\_\_DIR\_\_ . '/../vendor/autoload.php');

// 包含 Yii 类文件

require(\_\_DIR\_\_ . '/../vendor/yiisoft/yii2/Yii.php');

// 加载应用配置

$config = require(\_\_DIR\_\_ . '/../config/web.php');

// 创建、配置、运行一个应用

(new yii\web\Application($config))->run();

控制台应用

以下是一个控制台应用的入口脚本：

#!/usr/bin/env php

<?php

/\*\*

\* Yii console bootstrap file.

\*

\* @link http://www.yiiframework.com/

\* @copyright Copyright (c) 2008 Yii Software LLC

\* @license http://www.yiiframework.com/license/

\*/

defined('YII\_DEBUG') or define('YII\_DEBUG', true);

// 注册 Composer 自动加载器

require(\_\_DIR\_\_ . '/vendor/autoload.php');

// 包含 Yii 类文件

require(\_\_DIR\_\_ . '/vendor/yiisoft/yii2/Yii.php');

// 加载应用配置

$config = require(\_\_DIR\_\_ . '/config/console.php');

$application = new yii\console\Application($config);

$exitCode = $application->run();

exit($exitCode);

定义常量

入口脚本是定义全局常量的最好地方，Yii 支持以下三个常量：

YII\_DEBUG：标识应用是否运行在调试模式。当在调试模式下， 应用会保留更多日志信息，如果抛出异常，会显示详细的错误调用堆栈。 因此，调试模式主要适合在开发阶段使用，YII\_DEBUG 默认值为 false。

YII\_ENV：标识应用运行的环境。 YII\_ENV 默认值为 'prod'，表示应用运行在线上产品环境。

YII\_ENABLE\_ERROR\_HANDLER：标识是否启用 Yii 提供的错误处理， 默认为 true。

当定义一个常量时，通常使用类似如下代码来定义：

defined('YII\_DEBUG') or define('YII\_DEBUG', true);

上面的代码等同于:

if (!defined('YII\_DEBUG')) {

define('YII\_DEBUG', true);

}

显然第一段代码更加简洁易懂。

常量定义应该在入口脚本的开头，这样包含其他 PHP 文件时， 常量就能生效。

**一、应用主体**

应用主体是管理 Yii 应用系统整体结构和生命周期的对象。 每个Yii应用系统只能包含一个应用主体， 应用主体在 入口脚本 中创建并能通过表达式 \Yii::$app 全局范围内访问。

当我们说"一个应用"，它可能是一个应用主体对象，也可能是一个应用系统， 是根据上下文来决定[译：中文为避免歧义，Application翻译为应用主体]。

Yii有两种应用主体: 网页应用主体 and 控制台应用主体， 如名称所示，前者主要处理网页请求，后者处理控制台请求。

应用主体配置

如下所示，当 入口脚本 创建了一个应用主体， 它会加载一个 配置 文件并传给应用主体。

require(\_\_DIR\_\_ . '/../vendor/autoload.php');

require(\_\_DIR\_\_ . '/../vendor/yiisoft/yii2/Yii.php');

// 加载应用主体配置

$config = require(\_\_DIR\_\_ . '/../config/web.php');

// 实例化应用主体、配置应用主体

(new yii\web\Application($config))->run();

由于应用主体配置比较复杂， 一般保存在多个类似如上web.php的 配置文件 当中。

应用主体属性：

应用主体配置文件中有许多重要的属性要配置， 这些属性指定应用主体的运行环境。 比如，应用主体需要知道如何加载 控制器 ， 临时文件保存到哪儿等等。以下我们简述这些属性。

必要属性

在一个应用中，至少要配置2个属性:

yii\base\Application::id 和 yii\base\Application::basePath。

yii\base\Application::id 属性用来区分其他应用的唯一标识ID。 主要给程序使用。为了方便协作，最好使用数字作为应用主体ID， 但不强制要求为数字。

yii\base\Application::basePath 指定该应用的根目录。 根目录包含应用系统所有受保护的源代码。 在根目录下可以看到对应MVC设计模式的models, views, controllers等子目录。

可以使用路径或 路径别名 来在配置 yii\base\Application::basePath 属性。 两种格式所对应的目录都必须存在，否则系统会抛出一个异常。 系统会使用 realpath() 函数规范化配置的路径.

yii\base\Application::basePath 属性经常用于派生一些其他重要路径（如runtime路径）， 因此，系统预定义 @app 代表这个路径。 派生路径可以通过这个别名组成（如@app/runtime代表runtime的路径）。

重要属性

本小节所描述的属性通常需要设置， 因为不同的应用属性不同。

yii\base\Application::aliases

该属性允许你用一个数组定义多个 别名。 数组的key为别名名称，值为对应的路径。 例如：

[

'aliases' => [

'@name1' => 'path/to/path1',

'@name2' => 'path/to/path2',

],

]

使用这个属性来定义别名， 代替 Yii::setAlias() 方法来设置。

bootstrap

这个属性很实用，它允许你用数组指定启动阶段 bootstrapping process需要运行的组件。 比如，如果你希望一个 模块 自定义 URL 规则， 你可以将模块ID加入到bootstrap数组中。

属性中的每个组件需要指定以下一项:

应用 组件 ID.

模块 ID.

类名.

配置数组.

创建并返回一个组件的无名称函数.

例如：

[

'bootstrap' => [

// 应用组件ID或模块ID

'demo',

// 类名

'app\components\Profiler',

// 配置数组

[

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

'level' => 3,

],

// 无名称函数

function () {

return new app\components\Profiler();

}

],

]

如果模块ID和应用组件ID同名， 优先使用应用组件ID，如果你想用模块ID， 可以使用如下无名称函数返回模块ID。

[

function () {

return Yii::$app->getModule('user');

},

]

在启动阶段，每个组件都会实例化。 如果组件类实现接口 yii\base\BootstrapInterface, 也会调用 bootstrap() 方法。

举一个实际的例子， Basic Application Template 应用主体配置中， 开发环境下会在启动阶段运行 debug 和 gii 模块。

if (YII\_ENV\_DEV) {

// configuration adjustments for 'dev' environment

$config['bootstrap'][] = 'debug';

$config['modules']['debug'] = 'yii\debug\Module';

$config['bootstrap'][] = 'gii';

$config['modules']['gii'] = 'yii\gii\Module';

}

注: 启动太多的组件会降低系统性能， 因为每次请求都需要重新运行启动组件，因此谨慎配置启动组件。

catchAll

该属性仅 Web applications 网页应用支持。 它指定一个要处理所有用户请求的 控制器方法， 通常在维护模式下使用，同一个方法处理所有用户请求。

该配置为一个数组，第一项指定动作的路由， 剩下的数组项(key-value 成对)指定传递给动作的参数，例如：

[

'catchAll' => [

'offline/notice',

'param1' => 'value1',

'param2' => 'value2',

],

]

yii\base\Application::components

这是最重要的属性，它允许你注册多个在其他地方使用的应用组件。 例如

[

'components' => [

'cache' => [

'class' => 'yii\caching\FileCache',

],

'user' => [

'identityClass' => 'app\models\User',

'enableAutoLogin' => true,

],

],

]

每一个应用组件指定一个key-value对的数组，key代表组件ID， value代表组件类名或 配置。

在应用中可以任意注册组件， 并可以通过表达式 \Yii::$app->ComponentID 全局访问。

yii\base\Application::controllerMap

该属性允许你指定一个控制器ID到任意控制器类。 Yii遵循一个默认的 规则指定控制器ID到任意控制器类（如post对应app\controllers\PostController）。 通过配置这个属性，可以打破这个默认规则，在下面的例子中， account对应到app\controllers\UserController， article 对应到 app\controllers\PostController。

[

'controllerMap' => [

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

'article' => [

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

'enableCsrfValidation' => false,

],

],

]

数组的键代表控制器ID， 数组的值代表对应的类名。

controllerNamespace

该属性指定控制器类默认的命名空间，默认为app\controllers。 比如控制器ID为 post 默认对应 PostController （不带命名空间）， 类全名为 app\controllers\PostController。

控制器类文件可能放在这个命名空间对应目录的子目录下， 例如，控制器ID admin/post 对应的控制器类全名为 app\controllers\admin\PostController。

控制器类全面能被 自动加载，这点是非常重要的， 控制器类的实际命名空间对应这个属性， 否则，访问时你会收到"Page Not Found"[译：页面找不到]。

如果你想打破上述的规则， 可以配置 controllerMap 属性。

language

yii\base\Application::params

该属性为一个数组，指定可以全局访问的参数， 代替程序中硬编码的数字和字符， 应用中的参数定义到一个单独的文件并随时可以访问是一个好习惯。 例如用参数定义缩略图的长宽如下：

[

'params' => [

'thumbnail.size' => [128, 128],

],

]

然后简单的使用如下代码即可获取到你需要的长宽参数：

$size = \Yii::$app->params['thumbnail.size'];

$width = \Yii::$app->params['thumbnail.size'][0];

以后想修改缩略图长宽， 只需要修改该参数而不需要相关的代码。

yii\base\Application::timeZone

该属性提供一种方式修改PHP运行环境中的默认时区，配置该属性本质上就是调用PHP函数 date\_default\_timezone\_set()， 例如：

[

'timeZone' => 'America/Los\_Angeles',

]

yii\base\Application::version

该属性指定应用的版本，默认为'1.0'， 其他代码不使用的话可以不配置。

实用属性

本小节描述的属性不经常设置，通常使用系统默认值。 如果你想改变默认值，可以配置这些属性。

charset

该属性指定应用使用的字符集，默认值为 'UTF-8'， 绝大部分应用都在使用，除非已有的系统大量使用非unicode数据才需要更改该属性。

yii\base\Application::defaultRoute

该属性指定未配置的请求的响应 路由 规则， 路由规则可能包含模块ID，控制器ID，动作ID。 例如help, post/create, admin/post/create，如果动作ID没有指定， 会使用yii\base\Controller::$defaultAction中指定的默认值。

对于 Web applications 网页应用， 默认值为 'site' 对应 SiteController 控制器，并使用默认的动作。 因此你不带路由的访问应用，默认会显示 app\controllers\SiteController::actionIndex() 的结果。

对于 console applications 控制台应用， 默认值为 'help' 对应 yii\console\controllers\HelpController::actionIndex()。 因此，如果执行的命令不带参数，默认会显示帮助信息。

extensions

该属性用数组列表指定应用安装和使用的 扩展， 默认使用@vendor/yiisoft/extensions.php文件返回的数组。 当你使用 Composer 安装扩展，extensions.php 会被自动生成和维护更新。 所以大多数情况下，不需要配置该属性。

layout

该属性指定渲染 视图 默认使用的布局名字， 默认值为 'main' 对应布局路径下的 main.php 文件， 如果 布局路径 和 视图路径 都是默认值， 默认布局文件可以使用路径别名@app/views/layouts/main.php

如果不想设置默认布局文件，可以设置该属性为 false，这种做法比较罕见。

yii\base\Application::runtimePath

该属性指定临时文件如日志文件、缓存文件等保存路径， 默认值为带别名的 @app/runtime。

可以配置该属性为一个目录或者路径 别名， 注意应用运行时有对该路径的写入权限， 以及终端用户不能访问该路径因为临时文件可能包含一些敏感信息。

为了简化访问该路径，Yii预定义别名 @runtime 代表该路径。

yii\base\Application::viewPath

该路径指定视图文件的根目录，默认值为带别名的 @app/views， 可以配置它为一个目录或者路径 别名.

yii\base\Application::vendorPath

该属性指定 Composer 管理的供应商路径， 该路径包含应用使用的包括 Yii 框架在内的所有第三方库。 默认值为带别名的 @app/vendor 。

可以配置它为一个目录或者路径 别名，当你修改时， 务必修改对应的 Composer 配置。

为了简化访问该路径，Yii预定义别名 @vendor 代表该路径。

enableCoreCommands

该属性仅 console applications 控制台应用支持， 用来指定是否启用Yii中的核心命令，默认值为 true。

应用事件

应用在处理请求过程中会触发事件， 可以在配置文件配置事件处理代码，如下所示：

[

'on beforeRequest' => function ($event) {

// ...

},

]

on eventName

语法的用法在 Configurations 一节有详细描述.

另外，在应用主体实例化后，你可以在引导启动 阶段附加事件处理代码， 例如：

\Yii::$app->on(\yii\base\Application::EVENT\_BEFORE\_REQUEST, function ($event) {

// ...

});

EVENT\_BEFORE\_REQUEST

该事件在应用处理请求before之前，实际的事件名为 beforeRequest。

在事件触发前，应用主体已经实例化并配置好了， 所以通过事件机制将你的代码嵌入到请求处理过程中非常不错。 例如在事件处理中根据某些参数动态设置yii\base\Application::$language语言属性。

EVENT\_AFTER\_REQUEST

该事件在应用处理请求after之后但在返回响应before之前触发， 实际的事件名为afterRequest。

该事件触发时，请求已经被处理完， 可以做一些请求后处理或自定义响应。

注意 response 组件在发送响应给终端用户时也会触发一些事件， 这些事件都在本事件after之后触发。

yii\base\Application::EVENT\_BEFORE\_ACTION

该事件在每个 控制器动作 运行before之前会被触发， 实际的事件名为 beforeAction.

事件的参数为一个 yii\base\ActionEvent 实例， 事件处理中可以设置yii\base\ActionEvent::$isValid 为 false 停止运行后续动作， 例如：

[

'on beforeAction' => function ($event) {

if (some condition) {

$event->isValid = false;

} else {

}

},

]

注意 模块 和 控制器 都会触发 beforeAction 事件。 应用主体对象首先触发该事件，然后模块触发（如果存在模块），最后控制器触发。 任何一个事件处理中设置 yii\base\ActionEvent::$isValid 设置为 false 会停止触发后面的事件。

yii\base\Application::EVENT\_AFTER\_ACTION

该事件在每个 控制器动作 运行after之后会被触发， 实际的事件名为 afterAction.

该事件的参数为 yii\base\ActionEvent 实例， 通过yii\base\ActionEvent::$result属性， 事件处理可以访问和修改动作的结果。例如：

[

'on afterAction' => function ($event) {

if (some condition) {

// 修改 $event->result

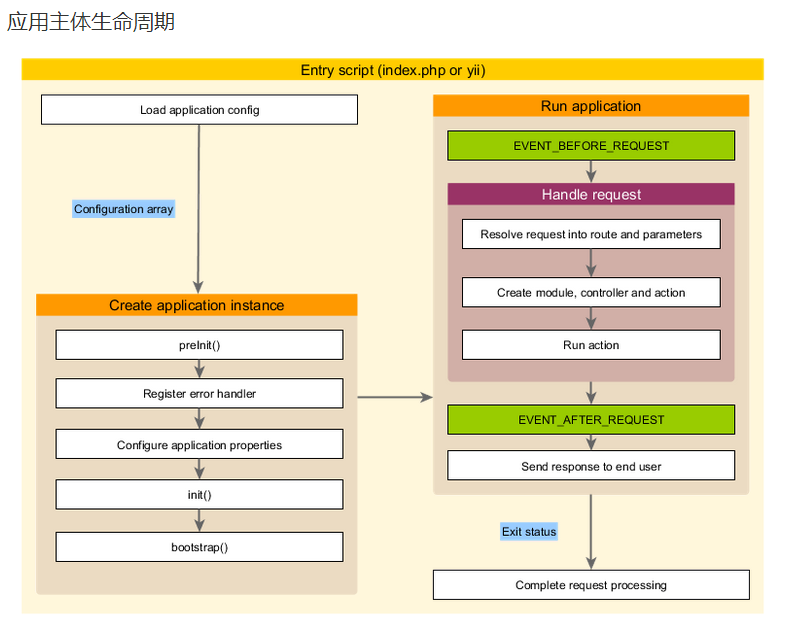
} else {

}

},

]

注意 模块 和 控制器 都会触发 afterAction 事件。 这些对象的触发顺序和 beforeAction 相反，也就是说，控制器最先触发， 然后是模块（如果有模块），最后为应用主体。



当运行 入口脚本 处理请求时， 应用主体会经历以下生命周期:

1、入口脚本加载应用主体配置数组。

2、入口脚本创建一个应用主体实例：

调用 preInit() 配置几个高级别应用主体属性， 比如yii\base\Application::basePath。

注册 yii\base\Application::errorHandler 错误处理方法.

配置应用主体属性.

调用 init() 初始化， 该函数会调用 bootstrap() 运行引导启动组件.

3、入口脚本调用 yii\base\Application::run() 运行应用主体:

触发 EVENT\_BEFORE\_REQUEST 事件。

处理请求：解析请求 路由 和相关参数； 创建路由指定的模块、控制器和动作对应的类，并运行动作。

触发 EVENT\_AFTER\_REQUEST 事件。

发送响应到终端用户.

4、入口脚本接收应用主体传来的退出状态并完成请求的处理。

**二、应用组件**

应用主体是服务定位器， 它部署一组提供各种不同功能的 应用组件 来处理请求。 例如，urlManager组件负责处理网页请求路由到对应的控制器。 db组件提供数据库相关服务等等。

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， 你可以通过如下表达式访问应用组件。

\Yii::$app->componentID

例如，可以使用 \Yii::$app->db 来获取到已注册到应用的 DB connection， 使用 \Yii::$app->cache 来获取到已注册到应用的 primary cache。

第一次使用以上表达式时候会创建应用组件实例， 后续再访问会返回此实例，无需再次创建。

应用组件可以是任意对象，可以在 应用主体配置配置 yii\base\Application::components 属性 . 例如：

[

'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;

},

],

]

补充: 请谨慎注册太多应用组件， 应用组件就像全局变量， 使用太多可能加大测试和维护的难度。 一般情况下可以在需要时再创建本地组件。

引导启动组件

上面提到一个应用组件只会在第一次访问时实例化， 如果处理请求过程没有访问的话就不实例化。 有时你想在每个请求处理过程都实例化某个组件即便它不会被访问， 可以将该组件ID加入到应用主体的 bootstrap 属性中。

例如, 如下的应用主体配置保证了 log 组件一直被加载。

[

'bootstrap' => [

// 将 log 组件 ID 加入引导让它始终载入

'log',

],

'components' => [

'log’ => [

// "log" 组件的配置

],

],

]

核心应用组件

Yii 定义了一组固定ID和默认配置的 核心 组件，例如 yii\web\Application::request 组件 用来收集用户请求并解析 路由； yii\base\Application::db 代表一个可以执行数据库操作的数据库连接。 通过这些组件，Yii应用主体能处理用户请求。

可以和普通应用组件一样配置和自定义它们。 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。

**三、控制器**

控制器是 MVC 模式中的一部分， 是继承yii\base\Controller类的对象，负责处理请求和生成响应。 具体来说，控制器从应用主体接管控制后会分析请求数据并传送到模型， 传送模型结果到视图， 最后生成输出响应信息。

动作

控制器由 动作 组成，它是执行终端用户请求的最基础的单元， 一个控制器可有一个或多个动作。

如下示例显示包含两个动作view and create 的控制器post：

namespace app\controllers;

use Yii;

use app\models\Post;

use yii\web\Controller;

use yii\web\NotFoundHttpException;

class PostController extends Controller

{

public function actionView($id)

{

$model = Post::findOne($id);

if ($model === null) {

throw new NotFoundHttpException;

}

return $this->render('view', [

'model' => $model,

]);

}

public function actionCreate()

{

$model = new Post;

if ($model->load(Yii::$app->request->post()) && $model->save()) {

return $this->redirect(['view', 'id' => $model->id]);

} else {

return $this->render('create', [

'model' => $model,

]);

}

}

}

在动作 view (定义为 actionView() 方法)中， 代码首先根据请求模型ID加载 模型， 如果加载成功，会渲染名称为view的视图并显示，否则会抛出一个异常。

在动作 create (定义为 actionCreate() 方法)中, 代码相似。 先将请求数据填入模型， 然后保存模型，如果两者都成功，会跳转到ID为新创建的模型的view动作， 否则显示提供用户输入的create视图。

路由

终端用户通过所谓的*路由*寻找到动作，路由是包含以下部分的字符串：

模型ID: 仅存在于控制器属于非应用的[模块](http://www.yiichina.com/doc/guide/2.0/structure-modules);

控制器ID: 同应用（或同模块如果为模块下的控制器） 下唯一标识控制器的字符串;

动作ID: 同控制器下唯一标识操作的字符串。

路由使用如下格式:

ControllerID/ActionID

如果属于模块下的控制器，使用如下格式：

ModuleID/ControllerID/ActionID

如果用户的请求地址为 http://hostname/index.php?r=site/index, 会执行site 控制器的index 动作。 更多关于处理路由的详情请参阅 [路由](http://www.yiichina.com/doc/guide/2.0/runtime-routing) 一节。

创建控制器

在[Web applications](http://www.yiichina.com/doc/api/2.0/yii-web-application)网页应用中，控制器应继承[yii\web\Controller](http://www.yiichina.com/doc/api/2.0/yii-web-controller) 或它的子类。 同理在[console applications](http://www.yiichina.com/doc/api/2.0/yii-console-application)控制台应用中，控制器继承[yii\console\Controller](http://www.yiichina.com/doc/api/2.0/yii-console-controller) 或它的子类。 如下代码定义一个 site 控制器:

namespace app\controllers;

use yii\web\Controller;

class SiteController extends Controller

{

}

控制器ID

通常情况下，控制器用来处理请求有关的资源类型， 因此控制器ID通常为和资源有关的名词。 例如使用article作为处理文章的控制器ID。

控制器ID应仅包含英文小写字母、数字、下划线、中横杠和正斜杠， 例如 article 和 post-comment 是真实的控制器 ID，article?, PostComment, admin\post不是控制器ID。

控制器Id可包含子目录前缀，例如 admin/article 代表 controller namespace控制器命名空间下 admin子目录中 article 控制器。 子目录前缀可为英文大小写字母、数字、下划线、正斜杠， 其中正斜杠用来区分多级子目录(如 panels/admin)。

控制器类命名

控制器ID遵循以下规则衍生控制器类名：

1. 将用正斜杠区分的每个单词第一个字母转为大写。注意如果控制器ID包含正斜杠， 只将最后的正斜杠后的部分第一个字母转为大写；
2. 去掉中横杠，将正斜杠替换为反斜杠;
3. 增加Controller后缀;
4. 在前面增加[controller namespace](http://www.yiichina.com/doc/api/2.0/yii-base-application#$controllerNamespace-detail)控制器命名空间.

下面为一些示例，假设[controller namespace](http://www.yiichina.com/doc/api/2.0/yii-base-application#$controllerNamespace-detail) 控制器命名空间为 app\controllers:

* article 对应 app\controllers\ArticleController;
* post-comment 对应 app\controllers\PostCommentController;
* admin/post-comment 对应 app\controllers\admin\PostCommentController;
* adminPanels/post-comment 对应 app\controllers\adminPanels\PostCommentController.

控制器类必须能被 [自动加载](http://www.yiichina.com/doc/guide/2.0/concept-autoloading)，所以在上面的例子中， 控制器article 类应在 [别名](http://www.yiichina.com/doc/guide/2.0/concept-aliases) 为@app/controllers/ArticleController.php的文件中定义， 控制器admin/post-comment应在@app/controllers/admin/PostCommentController.php文件中。

最后一个示例 admin/post-comment 表示你可以将控制器放在 [controller namespace](http://www.yiichina.com/doc/api/2.0/yii-base-application#$controllerNamespace-detail)控制器命名空间下的子目录中， 在你不想用 [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules) 的情况下给控制器分类，这种方式很有用。

控制器部署

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。

配置 [应用配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations) 中的[application configuration](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations)，如下所示：

[

'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,

],

],

]

默认控制器

每个应用有一个由yii\base\Application::defaultRoute属性指定的默认控制器；当请求没有指定 [路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#ids-routes)，该属性值作为路由使用。 对于[Web applications](http://www.yiichina.com/doc/api/2.0/yii-web-application)网页应用，它的值为 'site'， 对于 [console applications](http://www.yiichina.com/doc/api/2.0/yii-console-application)控制台应用，它的值为 help， 所以URL为 http://hostname/index.php 表示由 site 控制器来处理。

可以在 [应用配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations) 中修改默认控制器，如下所示：

[

'defaultRoute' => 'main',

]

创建动作

创建动作可简单地在控制器类中定义所谓的 操作方法 来完成，操作方法必须是以action开头的公有方法。 动作方法的返回值会作为响应数据发送给终端用户， 如下代码定义了两个动作 index 和 hello-world:

namespace app\controllers;

use yii\web\Controller;

class SiteController extends Controller

{

public function actionIndex()

{

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

}

public function actionHelloWorld()

{

return 'Hello World';

}

}

动作ID

动作通常是用来执行资源的特定操作，因此， 动作ID通常为动词，如view, update等。

动作ID应仅包含英文小写字母、数字、下划线和中横杠，操作ID中的中横杠用来分隔单词。 例如view, update2, comment-post是真实的动作 ID，view?, Update不是动作ID.

可通过两种方式创建动作ID，内联动作和独立动作. An inline action is 内联动作在控制器类中定义为方法；独立动作是继承[yii\base\Action](http://www.yiichina.com/doc/api/2.0/yii-base-action)或它的子类的类。 内联动作容易创建，在无需重用的情况下优先使用； 独立动作相反，主要用于多个控制器重用， 或重构为[扩展](http://www.yiichina.com/doc/guide/2.0/structure-extensions)。

内联动作

内联动作指的是根据我们刚描述的操作方法。

动作方法的名字是根据操作ID遵循如下规则衍生：

* 将每个单词的第一个字母转为大写;
* 去掉中横杠;
* 增加action前缀.

例如index 转成 actionIndex, hello-world 转成 actionHelloWorld。

**Note:** 动作方法的名字*大小写敏感*，如果方法名称为ActionIndex不会认为是操作方法， 所以请求index动作会返回一个异常， 也要注意动作方法必须是公有的， 私有或者受保护的方法不能定义成内联动作。

因为容易创建，内联动作是最常用的操作， 但是如果你计划在不同地方重用相同的动作， 或者你想重新分配一个动作，需要考虑定义它为*独立动作*。

独立动作

独立动作通过继承[yii\base\Action](http://www.yiichina.com/doc/api/2.0/yii-base-action)或它的子类来定义。 例如Yii发布的[yii\web\ViewAction](http://www.yiichina.com/doc/api/2.0/yii-web-viewaction)和[yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction) 都是独立动作。

要使用独立动作，需要通过控制器中覆盖[yii\base\Controller::actions()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#actions%28%29-detail)方法在action map中申明， 如下例所示：

public function actions()

{

return [

// 用类来申明"error" 动作

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

// 用配置数组申明 "view" 动作

'view' => [

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

'viewPrefix' => '',

],

];

}

如上所示， actions() 方法返回键为动作ID、值为对应操作类名或数组configurations 的数组。 和内联动作不同，独立动作ID可包含任意字符， 只要在actions() 方法中申明.

为创建一个独立动作类，需要继承yii\base\Action 或它的子类，并实现公有的名称为run()的方法, run() 方法的角色和动作方法类似，例如：

<?php

namespace app\components;

use yii\base\Action;

class HelloWorldAction extends Action

{

public function run()

{

return "Hello World";

}

}

动作结果

动作方法或独立操作的run()方法的返回值非常重要， 它表示对应动作结果。

返回值可为 [响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses) 对象，作为响应发送给终端用户。

* 对于[Web applications](http://www.yiichina.com/doc/api/2.0/yii-web-application)网页应用，返回值可为任意数据, 它赋值给[yii\web\Response::$data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail)， 最终转换为字符串来展示响应内容。
* 对于[console applications](http://www.yiichina.com/doc/api/2.0/yii-console-application)控制台应用，返回值可为整数， 表示命令行下执行的 yii\console\Response::exitStatus 退出状态。

在上面的例子中，动作结果都为字符串，作为响应数据发送给终端用户， 下例显示一个动作通过 返回响应对象（因为[redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-controller#redirect%28%29-detail)方法返回一个响应对象） 可将用户浏览器跳转到新的URL。

public function actionForward()

{

// 用户浏览器跳转到 http://example.com

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

}

动作参数

内联动作的操作方法和独立动作的 run() 方法可以带参数，称为*动作参数*。 参数值从请求中获取，对于[Web applications](http://www.yiichina.com/doc/api/2.0/yii-web-application)网页应用， 每个动作参数的值从$\_GET中获得，参数名作为键； 对于[console applications](http://www.yiichina.com/doc/api/2.0/yii-console-application)控制台应用, 动作参数对应命令行参数。

如下例，动作view (内联动作) 申明了两个参数 $id 和 $version。

namespace app\controllers;

use yii\web\Controller;

class PostController extends Controller

{

public function actionView($id, $version = null)

{

// ...

}

}

动作参数会被不同的参数填入，如下所示：

* http://hostname/index.php?r=post/view&id=123: $id 会填入'123'， $version 仍为 null 空因为没有version请求参数;
* http://hostname/index.php?r=post/view&id=123&version=2: $id 和 $version 分别填入 '123' 和 '2'`；
* http://hostname/index.php?r=post/view: 会抛出[yii\web\BadRequestHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-badrequesthttpexception) 异常 因为请求没有提供参数给必须赋值参数$id；
* http://hostname/index.php?r=post/view&id[]=123: 会抛出[yii\web\BadRequestHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-badrequesthttpexception) 异常 因为$id 参数收到数字值 ['123']而不是字符串.

如果想让动作参数接收数组值，需要指定$id为array，如下所示：

public function actionView(array $id, $version = null)

{

// ...

}

现在如果请求为 http://hostname/index.php?r=post/view&id[]=123, 参数 $id 会使用数组值['123']， 如果请求为 http://hostname/index.php?r=post/view&id=123， 参数 $id 会获取相同数组值，因为无类型的'123'会自动转成数组。

上述例子主要描述网页应用的动作参数，对于控制台应用， 更多详情请参阅[控制台命令](http://www.yiichina.com/doc/guide/2.0/tutorial-console)。

默认动作

每个控制器都有一个由 [yii\base\Controller::$defaultAction](http://www.yiichina.com/doc/api/2.0/yii-base-controller#$defaultAction-detail) 属性指定的默认动作， 当[路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#ids-routes) 只包含控制器ID， 会使用所请求的控制器的默认动作。

默认动作默认为 index，如果想修改默认动作， 只需简单地在控制器类中覆盖这个属性，如下所示：

namespace app\controllers;

use yii\web\Controller;

class SiteController extends Controller

{

public $defaultAction = 'home';

public function actionHome()

{

return $this->render('home');

}

}

控制器生命周期

处理一个请求时， [应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications) 会根据请求[路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes)创建一个控制器， 控制器经过以下生命周期来完成请求：

* 在控制器创建和配置后，yii\base\Controller::init() 方法会被调用。
* 控制器根据请求动作ID创建一个操作对象:
* 如果动作ID没有指定，会使用[default action ID](http://www.yiichina.com/doc/api/2.0/yii-base-controller#$defaultAction-detail)默认操作ID；
* 如果在[action map](http://www.yiichina.com/doc/api/2.0/yii-base-controller#actions%28%29-detail)找到动作ID， 会创建一个独立动作；
* 如果动作ID对应操作方法，会创建一个内联操作；
* 否则会抛出[yii\base\InvalidRouteException](http://www.yiichina.com/doc/api/2.0/yii-base-invalidrouteexception)异常。
* 控制器按顺序调用应用主体、模块（如果控制器属于模块）、 控制器的 beforeAction() 方法；
* 如果任意一个调用返回false，后面未调用的beforeAction()会跳过并且动作执行会被取消； action execution will be cancelled.
* 默认情况下每个 beforeAction() 方法会触发一个 beforeAction 事件，在事件中你可以追加事件处理动作；
* 控制器执行动作:
* 请求数据解析和填入到动作参数；
* 控制器按顺序调用控制器、模块（如果控制器属于模块）、 应用主体的 afterAction() 方法；
* 默认情况下每个 afterAction() 方法会触发一个 afterAction 事件，在事件中你可以追加事件处理动作；
* 应用主体获取动作结果并赋值给[响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses).

最佳实践

在设计良好的应用中，控制器很精练，包含的动作代码简短； 如果你的控制器很复杂，通常意味着需要重构， 转移一些代码到其他类中。

归纳起来，控制器

* 可访问 [请求](http://www.yiichina.com/doc/guide/2.0/runtime-requests) 数据;
* 可根据请求数据调用 [模型](http://www.yiichina.com/doc/guide/2.0/structure-models) 的方法和其他服务组件;
* 可使用 [视图](http://www.yiichina.com/doc/guide/2.0/structure-views) 构造响应;
* 不应处理应被[模型](http://www.yiichina.com/doc/guide/2.0/structure-models)处理的请求数据;
* 应避免嵌入HTML或其他展示代码，这些代码最好在 [视图](http://www.yiichina.com/doc/guide/2.0/structure-views)中处理.

**四、模型**

模型是 [MVC](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) 模式中的一部分， 是代表业务数据、规则和逻辑的对象。

可通过继承 [yii\base\Model](http://www.yiichina.com/doc/api/2.0/yii-base-model) 或它的子类定义模型类， 基类[yii\base\Model](http://www.yiichina.com/doc/api/2.0/yii-base-model)支持许多实用的特性：

* [属性](http://www.yiichina.com/doc/guide/2.0/structure-models#attributes): 代表可像普通类属性或数组 一样被访问的业务数据;
* [属性标签](http://www.yiichina.com/doc/guide/2.0/structure-models#attribute-labels): 指定属性显示出来的标签;
* [块赋值](http://www.yiichina.com/doc/guide/2.0/structure-models#massive-assignment): 支持一步给许多属性赋值;
* [验证规则](http://www.yiichina.com/doc/guide/2.0/structure-models#validation-rules): 确保输入数据符合所申明的验证规则;
* [数据导出](http://www.yiichina.com/doc/guide/2.0/structure-models#data-exporting): 允许模型数据导出为自定义格式的数组。

Model 类也是更多高级模型如[Active Record 活动记录](http://www.yiichina.com/doc/guide/2.0/db-active-record)的基类， 更多关于这些高级模型的详情请参考相关手册。

**Info:** 模型并不强制一定要继承[yii\base\Model](http://www.yiichina.com/doc/api/2.0/yii-base-model)，但是由于很多组件支持[yii\base\Model](http://www.yiichina.com/doc/api/2.0/yii-base-model)， 最好使用它做为模型基类。

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%28%29-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";

}

定义属性

默认情况下你的模型类直接从[yii\base\Model](http://www.yiichina.com/doc/api/2.0/yii-base-model)继承，所有 *non-static public非静态公有* 成员变量都是属性。 例如，下述ContactForm 模型类有四个属性name, email, subject and body， ContactForm 模型用来代表从HTML表单获取的输入数据。

namespace app\models;

use yii\base\Model;

class ContactForm extends Model

{

public $name;

public $email;

public $subject;

public $body;

}

另一种方式是可覆盖 [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes%28%29-detail) 来定义属性， 该方法返回模型的属性名。 例如 [yii\db\ActiveRecord](http://www.yiichina.com/doc/api/2.0/yii-db-activerecord) 返回对应数据表列名作为它的属性名， 注意可能需要覆盖魔术方法如\_\_get(), \_\_set()使属性像普通对象属性被访问。

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%28%29-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%28%29-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%28%29-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%28%29-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。

**注意:** 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。

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%28%29-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'],

];

}

}

**Info:** 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%28%29-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)。

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%28%29-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%28%29-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%28%29-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。

块赋值

块赋值只用一行代码将用户所有输入填充到一个模型，非常方便， 它直接将输入数据对应填充到 [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes%28%29-detail) 属性。 以下两段代码效果是相同的， 都是将终端用户输入的表单数据赋值到 ContactForm 模型的属性， 明显地前一段块赋值的代码比后一段代码简洁且不易出错。

$model = new \app\models\ContactForm;

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

$model = new \app\models\ContactForm;

$data = \Yii::$app->request->post('ContactForm', []);

$model->name = isset($data['name']) ? $data['name'] : null;

$model->email = isset($data['email']) ? $data['email'] : null;

$model->subject = isset($data['subject']) ? $data['subject'] : null;

$model->body = isset($data['body']) ? $data['body'] : null;

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%28%29-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'],

];

}

**Info:** 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%28%29-detail)的实现 会返回[yii\base\Model::rules()](http://www.yiichina.com/doc/api/2.0/yii-base-model#rules%28%29-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'],

];

}

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%28%29-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.

数据导出

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， 第一步，模型转换成数组；第二步，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%28%29-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%28%29-detail) 属性会返回 所有 [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes%28%29-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%28%29-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).

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%28%29-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;

}

Warning: 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。

最佳实践

模型是代表业务数据、规则和逻辑的中心地方，通常在很多地方重用， 在一个设计良好的应用中， 模型通常比[控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers)代码多。

归纳起来，模型

* 可包含属性来展示业务数据;
* 可包含验证规则确保数据有效和完整;
* 可包含方法实现业务逻辑;
* 不应直接访问请求，session和其他环境数据， 这些数据应该由[控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers)传入到模型;
* 应避免嵌入HTML或其他展示代码，这些代码最好在 [视图](http://www.yiichina.com/doc/guide/2.0/structure-views)中处理;
* 单个模型中避免太多的 [场景](http://www.yiichina.com/doc/guide/2.0/structure-models#scenarios).

在开发大型复杂系统时应经常考虑最后一条建议， 在这些系统中，模型会很大并在很多地方使用，因此会包含需要规则集和业务逻辑， 最后维护这些模型代码成为一个噩梦， 因为一个简单修改会影响好多地方， 为确保模型好维护，最好使用以下策略：

* 定义可被多个 [应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications) 或 [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules) 共享的模型基类集合。 这些模型类应包含通用的最小规则集合和逻辑。
* 在每个使用模型的 [应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications) 或 [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules)中， 通过继承对应的模型基类来定义具体的模型类， 具体模型类包含应用主体或模块指定的规则和逻辑。

例如，在[高级应用模板](http://www.yiichina.com/doc/guide/2.0/tutorial-advanced-app)， 你可以定义一个模型基类common\models\Post， 然后在前台应用中，定义并使用一个继承common\models\Post的具体模型类frontend\models\Post， 在后台应用中可以类似地定义backend\models\Post。 通过这种策略，你清楚frontend\models\Post只对应前台应用， 如果你修改它，就无需担忧修改会影响后台应用。

**五、视图**

视图是 [MVC](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) 模式中的一部分。 它是展示数据到终端用户的代码，在网页应用中， 根据*视图模板*来创建视图，视图模板为PHP脚本文件， 主要包含HTML代码和展示类PHP代码，通过[view](http://www.yiichina.com/doc/api/2.0/yii-web-view)应用组件来管理， 该组件主要提供通用方法帮助视图构造和渲染，简单起见， 我们称视图模板或视图模板文件为视图。

创建视图

如前所述，视图为包含HTML和PHP代码的PHP脚本，如下代码为一个登录表单的视图， 可看到PHP代码用来生成动态内容如页面标题和表单， HTML代码把它组织成一个漂亮的HTML页面。

<?php

use yii\helpers\Html;

use yii\widgets\ActiveForm;

/\* @var $this yii\web\View \*/

/\* @var $form yii\widgets\ActiveForm \*/

/\* @var $model app\models\LoginForm \*/

$this->title = 'Login';

?>

<h1><?= Html::encode($this->title) ?></h1>

<p>Please fill out the following fields to login:</p>

<?php $form = ActiveForm::begin(); ?>

<?= $form->field($model, 'username') ?>

<?= $form->field($model, 'password')->passwordInput() ?>

<?= Html::submitButton('Login') ?>

<?php ActiveForm::end(); ?>

在视图中，可访问 $this 指向 [view component](http://www.yiichina.com/doc/api/2.0/yii-web-view) 来管理和渲染这个视图文件。

除了 $this之外，上述示例中的视图有其他预定义变量如 $model， 这些变量代表从[控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers)或 其他触发[视图渲染](http://www.yiichina.com/doc/guide/2.0/structure-views#rendering-views)的对象 传入 到视图的数据。

技巧: 将预定义变量列到视图文件头部注释处，这样可被IDE编辑器识别， 也是生成视图文档的好方法。

安全

当创建生成HTML页面的视图时， 在显示之前将用户输入数据进行转码和过滤非常重要， 否则，你的应用可能会被[跨站脚本](http://en.wikipedia.org/wiki/Cross-site_scripting) 攻击。

要显示纯文本，先调用 yii\helpers\Html::encode() 进行转码， 例如如下代码将用户名在显示前先转码：

<?php

use yii\helpers\Html;

?>

<div class="username">

<?= Html::encode($user->name) ?>

</div>

要显示HTML内容，先调用 [yii\helpers\HtmlPurifier](http://www.yiichina.com/doc/api/2.0/yii-helpers-htmlpurifier) 过滤内容， 例如如下代码将提交内容在显示前先过滤：

<?php

use yii\helpers\HtmlPurifier;

?>

<div class="post">

<?= HtmlPurifier::process($post->text) ?>

</div>

技巧：HTMLPurifier在保证输出数据安全上做的不错，但性能不佳，如果你的应用需要高性能可考虑 [缓存](http://www.yiichina.com/doc/guide/2.0/caching-overview) 过滤后的结果。

组织视图

与 [控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 和 [模型](http://www.yiichina.com/doc/guide/2.0/structure-models) 类似，在组织视图上有一些约定：

* 控制器渲染的视图文件默认放在 @app/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), 例如控制器类为 PostController，视图文件目录应为 @app/views/post， 控制器类 PostCommentController对应的目录为 @app/views/post-comment， 如果是模块中的控制器，目录应为 yii\base\Module::basePath 模块目录下的 views/ControllerID 目录；
* 对于 [小部件](http://www.yiichina.com/doc/guide/2.0/structure-widgets) 渲染的视图文件默认放在 WidgetPath/views 目录， 其中 WidgetPath 代表小部件类文件所在的目录；
* 对于其他对象渲染的视图文件，建议遵循和小部件相似的规则。

可覆盖控制器或小部件的 [yii\base\ViewContextInterface::getViewPath()](http://www.yiichina.com/doc/api/2.0/yii-base-viewcontextinterface#getViewPath%28%29-detail) 方法来自定义视图文件默认目录。

渲染视图

可在 [控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers), [小部件](http://www.yiichina.com/doc/guide/2.0/structure-widgets), 或其他地方调用渲染视图方法来渲染视图， 该方法类似以下格式：

/\*\*

\* @param string $view 视图名或文件路径，由实际的渲染方法决定

\* @param array $params 传递给视图的数据

\* @return string 渲染结果

\*/

methodName($view, $params = [])

控制器中渲染

在 [控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 中，可调用以下控制器方法来渲染视图：

* [render()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#render%28%29-detail): 渲染一个 [视图名](http://www.yiichina.com/doc/guide/2.0/structure-views#named-views) 并使用一个 [布局](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts) 返回到渲染结果。
* [renderPartial()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#renderPartial%28%29-detail): 渲染一个 [视图名](http://www.yiichina.com/doc/guide/2.0/structure-views#named-views) 并且不使用布局。
* [renderAjax()](http://www.yiichina.com/doc/api/2.0/yii-web-controller#renderAjax%28%29-detail): 渲染一个 [视图名](http://www.yiichina.com/doc/guide/2.0/structure-views#named-views) 并且不使用布局， 并注入所有注册的JS/CSS脚本和文件，通常使用在响应AJAX网页请求的情况下。
* [别名](http://www.yiichina.com/doc/guide/2.0/concept-aliases)下的视图文件。
* [renderContent()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#renderContent%28%29-detail): renders a static string by embedding it into the currently applicable [layout](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts). This method is available since version 2.0.1.

例如：

namespace app\controllers;

use Yii;

use app\models\Post;

use yii\web\Controller;

use yii\web\NotFoundHttpException;

class PostController extends Controller

{

public function actionView($id)

{

$model = Post::findOne($id);

if ($model === null) {

throw new NotFoundHttpException;

}

// 渲染一个名称为"view"的视图并使用布局

return $this->render('view', [

'model' => $model,

]);

}

}

小部件中渲染

在 [小部件](http://www.yiichina.com/doc/guide/2.0/structure-widgets) 中，可调用以下小部件方法来渲染视图：

* [render()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#render%28%29-detail): 渲染一个 [视图名](http://www.yiichina.com/doc/guide/2.0/structure-views#named-views).
* [别名](http://www.yiichina.com/doc/guide/2.0/concept-aliases)下的视图文件。

例如：

namespace app\components;

use yii\base\Widget;

use yii\helpers\Html;

class ListWidget extends Widget

{

public $items = [];

public function run()

{

// 渲染一个名为 "list" 的视图

return $this->render('list', [

'items' => $this->items,

]);

}

}

视图中渲染

可以在视图中渲染另一个视图，可以调用[view component](http://www.yiichina.com/doc/api/2.0/yii-base-view)视图组件提供的以下方法：

* [render()](http://www.yiichina.com/doc/api/2.0/yii-base-view#render%28%29-detail): 渲染一个 [视图名](http://www.yiichina.com/doc/guide/2.0/structure-views#named-views).
* [renderAjax()](http://www.yiichina.com/doc/api/2.0/yii-web-view#renderAjax%28%29-detail): 渲染一个 [视图名](http://www.yiichina.com/doc/guide/2.0/structure-views#named-views) 并注入所有注册的JS/CSS脚本和文件，通常使用在响应AJAX网页请求的情况下。
* [别名](http://www.yiichina.com/doc/guide/2.0/concept-aliases)下的视图文件。

例如，视图中的如下代码会渲染该视图所在目录下的 \_overview.php 视图文件， 记住视图中 $this 对应 [view](http://www.yiichina.com/doc/api/2.0/yii-base-view) 组件:

<?= $this->render('\_overview') ?>、

其他地方渲染

在任何地方都可以通过表达式 Yii::$app->view 访问 [view](http://www.yiichina.com/doc/api/2.0/yii-base-view) 应用组件， 调用它的如前所述的方法渲染视图，例如：

// 显示视图文件 "@app/views/site/license.php"

echo \Yii::$app->view->renderFile('@app/views/site/license.php');

视图名

渲染视图时，可指定一个视图名或视图文件路径/别名，大多数情况下使用前者因为前者简洁灵活， 我们称用名字的视图为 *视图名*.

视图名可以依据以下规则到对应的视图文件路径：

* 视图名可省略文件扩展名，这种情况下使用 .php 作为扩展， 视图名 about 对应到 about.php 文件名；
* 视图名以双斜杠 // 开头，对应的视图文件路径为 @app/views/ViewName， 也就是说视图文件在 yii\base\Application::viewPath 路径下找， 例如 //site/about 对应到 @app/views/site/about.php。
* 视图名以单斜杠/开始，视图文件路径以当前使用[模块](http://www.yiichina.com/doc/guide/2.0/structure-modules) 的yii\base\Module::viewPath开始， 如果不存在模块，使用@app/views/ViewName开始， 例如，如果当前模块为user， /user/create 对应成 @app/modules/user/views/user/create.php, 如果不在模块中，/user/create对应@app/views/user/create.php。
* 如果 [context](http://www.yiichina.com/doc/api/2.0/yii-base-view#$context-detail) 渲染视图 并且上下文实现了 [yii\base\ViewContextInterface](http://www.yiichina.com/doc/api/2.0/yii-base-viewcontextinterface), 视图文件路径由上下文的 [view path](http://www.yiichina.com/doc/api/2.0/yii-base-viewcontextinterface#getViewPath%28%29-detail) 开始， 这种主要用在控制器和小部件中渲染视图，例如 如果上下文为控制器SiteController，site/about 对应到 @app/views/site/about.php。
* 如果视图渲染另一个视图，包含另一个视图文件的目录以当前视图的文件路径开始， 例如被视图@app/views/post/index.php 渲染的 item 对应到 @app/views/post/item。

根据以上规则，在控制器中 app\controllers\PostController 调用 $this->render('view')， 实际上渲染 @app/views/post/view.php 视图文件，当在该视图文件中调用 $this->render('\_overview') 会渲染 @app/views/post/\_overview.php 视图文件。

视图中访问数据

在视图中有两种方式访问数据：推送和拉取。

推送方式是通过视图渲染方法的第二个参数传递数据， 数据格式应为名称-值的数组， 视图渲染时，调用PHP extract() 方法将该数组转换为视图可访问的变量。 例如，如下控制器的渲染视图代码推送2个变量到 report 视图：$foo = 1 和 $bar = 2。

echo $this->render('report', [

'foo' => 1,

'bar' => 2,

]);

拉取方式可让视图从[view component](http://www.yiichina.com/doc/api/2.0/yii-base-view)视图组件或其他对象中主动获得数据(如Yii::$app)， 在视图中使用如下表达式$this->context可获取到控制器ID， 可让你在report视图中获取控制器的任意属性或方法， 如以下代码获取控制器ID。

The controller ID is: <?= $this->context->id ?>

推送方式让视图更少依赖上下文对象，是视图获取数据优先使用方式， 缺点是需要手动构建数组，有些繁琐， 在不同地方渲染时容易出错。

视图间共享数据

[view component](http://www.yiichina.com/doc/api/2.0/yii-base-view)视图组件提供[params](http://www.yiichina.com/doc/api/2.0/yii-base-view#$params-detail)参数 属性来让不同视图共享数据。

例如在about视图中，可使用如下代码指定当前 breadcrumbs的当前部分。

$this->params['breadcrumbs'][] = 'About Us';

在[布局](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts)文件（也是一个视图）中，可使用依次加入到[params](http://www.yiichina.com/doc/api/2.0/yii-base-view#$params-detail)数组的值来 生成显示breadcrumbs:

<?= yii\widgets\Breadcrumbs::widget([

'links' => isset($this->params['breadcrumbs']) ? $this->params['breadcrumbs'] : [],

]) ?>

布局

布局是一种特殊的视图，代表多个视图的公共部分， 例如，大多数Web应用共享相同的页头和页尾， 在每个视图中重复相同的页头和页尾，更好的方式是将这些公共放到一个布局中， 渲染内容视图后在合适的地方嵌入到布局中。

创建布局

由于布局也是视图，它可像普通视图一样创建，布局默认存储在@app/views/layouts路径下， [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules)中使用的布局应存储在 yii\base\Module::basePath模块目录 下的views/layouts路径下，可配置yii\base\Module::layoutPath 来自定义应用或模块的布局默认路径。

如下示例为一个布局大致内容，注意作为示例，简化了很多代码， 在实际中，你可能想添加更多内容，如头部标签，主菜单等。

<?php

use yii\helpers\Html;

/\* @var $this yii\web\View \*/

/\* @var $content string 字符串 \*/

?>

<?php $this->beginPage() ?>

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8"/>

<?= Html::csrfMetaTags() ?>

<title><?= Html::encode($this->title) ?></title>

<?php $this->head() ?>

</head>

<body>

<?php $this->beginBody() ?>

<header>My Company</header>

<?= $content ?>

<footer>&copy; 2014 by My Company</footer>

<?php $this->endBody() ?>

</body>

</html>

<?php $this->endPage() ?>

如上所示，布局生成每个页面通用的HTML标签，在<body>标签中，打印$content变量， $content变量代表当[yii\base\Controller::render()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#render%28%29-detail) 控制器渲染方法调用时传递到布局的内容视图渲染结果。

大多数视图应调用上述代码中的如下方法， 这些方法触发关于渲染过程的事件， 这样其他地方注册的脚本和标签会添加到这些方法调用的地方。

* 它触发表明页面开始的 [EVENT\_BEGIN\_PAGE](http://www.yiichina.com/doc/api/2.0/yii-base-view#EVENT_BEGIN_PAGE-detail) 事件。
* 它触发表明页面结尾的 [EVENT\_END\_PAGE](http://www.yiichina.com/doc/api/2.0/yii-base-view#EVENT_END_PAGE-detail) 时间。
* 它生成一个占位符，在页面渲染结束时会被注册的头部HTML代码 （如，link标签, meta标签）替换。
* 它触发 [EVENT\_BEGIN\_BODY](http://www.yiichina.com/doc/api/2.0/yii-web-view#EVENT_BEGIN_BODY-detail) 事件并生成一个占位符， 会被注册的HTML代码（如JavaScript）在页面主体开始处替换。
* 它触发 [EVENT\_END\_BODY](http://www.yiichina.com/doc/api/2.0/yii-web-view#EVENT_END_BODY-detail) 事件并生成一个占位符， 会被注册的HTML代码（如JavaScript）在页面主体结尾处替换。

布局中访问数据

在布局中可访问两个预定义变量：$this 和 $content， 前者对应和普通视图类似的[view](http://www.yiichina.com/doc/api/2.0/yii-base-view) 视图组件 后者包含调用[render()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#render%28%29-detail)方法渲染内容视图的结果。

如果想在布局中访问其他数据，必须使用[视图中访问数据](http://www.yiichina.com/doc/guide/2.0/structure-views#accessing-data-in-views)一节介绍的拉取方式， 如果想从内容视图中传递数据到布局， 可使用[视图间共享数据](http://www.yiichina.com/doc/guide/2.0/structure-views#sharing-data-among-views)一节中的方法。

使用布局

如[控制器中渲染](http://www.yiichina.com/doc/guide/2.0/structure-views#rendering-in-controllers)一节描述，当控制器调用[render()](http://www.yiichina.com/doc/api/2.0/yii-base-controller#render%28%29-detail) 方法渲染视图时，会同时使用布局到渲染结果中， 默认会使用@app/views/layouts/main.php布局文件。

可配置[yii\base\Application::$layout](http://www.yiichina.com/doc/api/2.0/yii-base-application#$layout-detail) 或 [yii\base\Controller::$layout](http://www.yiichina.com/doc/api/2.0/yii-base-controller#$layout-detail) 使用其他布局文件， 前者管理所有控制器的布局，后者覆盖前者来控制单个控制器布局。 例如，如下代码使 post 控制器渲染视图时使用 @app/views/layouts/post.php 作为布局文件， 假如 layout 属性没改变， 控制器默认使用 @app/views/layouts/main.php 作为布局文件。

namespace app\controllers;

use yii\web\Controller;

class PostController extends Controller

{

public $layout = 'post';

// ...

}

对于模块中的控制器，可配置模块的 [layout](http://www.yiichina.com/doc/api/2.0/yii-base-module#$layout-detail) 属性指定布局文件应用到模块的所有控制器。

由于layout 可在不同层级（控制器、模块，应用）配置， 在幕后Yii使用两步来决定控制器实际使用的布局。

第一步，它决定布局的值和上下文模块：

* 如果控制器的 [yii\base\Controller::$layout](http://www.yiichina.com/doc/api/2.0/yii-base-controller#$layout-detail) 属性不为空null，使用它作为布局的值， 控制器的 [module](http://www.yiichina.com/doc/api/2.0/yii-base-controller#$module-detail)模块 作为上下文模块。
* 如果 [layout](http://www.yiichina.com/doc/api/2.0/yii-base-controller#$layout-detail) 为空，从控制器的祖先模块（包括应用） 开始找 第一个[layout](http://www.yiichina.com/doc/api/2.0/yii-base-module#$layout-detail) 属性不为空的模块，使用该模块作为上下文模块， 并将它的[layout](http://www.yiichina.com/doc/api/2.0/yii-base-module#$layout-detail) 的值作为布局的值， 如果都没有找到，表示不使用布局。

第二步，它决定第一步中布局的值和上下文模块对应到实际的布局文件， 布局的值可为：

* 路径别名 (如 @app/views/layouts/main).
* 绝对路径 (如 /main): 布局的值以斜杠开始，在应用的[[yii\base\Application::layoutPath|layout path] 布局路径 中查找实际的布局文件， 布局路径默认为 @app/views/layouts。
* 相对路径 (如 main): 在上下文模块的yii\base\Module::layoutPath 布局路径中查找实际的布局文件， 布局路径默认为yii\base\Module::basePath模块目录下的views/layouts 目录。
* 布尔值 false: 不使用布局。

布局的值没有包含文件扩展名，默认使用 .php作为扩展名。

嵌套布局

有时候你想嵌套一个布局到另一个，例如，在Web站点不同地方，想使用不同的布局， 同时这些布局共享相同的生成全局HTML5页面结构的基本布局，可以在子布局中调用 [beginContent()](http://www.yiichina.com/doc/api/2.0/yii-base-view#beginContent%28%29-detail) 和[endContent()](http://www.yiichina.com/doc/api/2.0/yii-base-view#endContent%28%29-detail) 方法，如下所示：

<?php $this->beginContent('@app/views/layouts/base.php'); ?>

...child layout content here...

<?php $this->endContent(); ?>

如上所示，子布局内容应在 [beginContent()](http://www.yiichina.com/doc/api/2.0/yii-base-view#beginContent%28%29-detail) 和 [endContent()](http://www.yiichina.com/doc/api/2.0/yii-base-view#endContent%28%29-detail) 方法之间，传给 [beginContent()](http://www.yiichina.com/doc/api/2.0/yii-base-view#beginContent%28%29-detail) 的参数指定父布局，父布局可为布局文件或别名。

使用以上方式可多层嵌套布局。

使用数据块

数据块可以在一个地方指定视图内容在另一个地方显示，通常和布局一起使用， 例如，可在内容视图中定义数据块在布局中显示它。

调用 [beginBlock()](http://www.yiichina.com/doc/api/2.0/yii-base-view#beginBlock%28%29-detail) 和 [endBlock()](http://www.yiichina.com/doc/api/2.0/yii-base-view#endBlock%28%29-detail) 来定义数据块， 使用 $view->blocks[$blockID] 访问该数据块， 其中 $blockID 为定义数据块时指定的唯一标识ID。

如下实例显示如何在内容视图中使用数据块让布局使用。

首先，在内容视图中定一个或多个数据块：

...

<?php $this->beginBlock('block1'); ?>

...content of block1...

<?php $this->endBlock(); ?>

...

<?php $this->beginBlock('block3'); ?>

...content of block3...

<?php $this->endBlock(); ?>

然后，在布局视图中，数据块可用的话会渲染数据块， 如果数据未定义则显示一些默认内容。

...

<?php if (isset($this->blocks['block1'])): ?>

<?= $this->blocks['block1'] ?>

<?php else: ?>

... default content for block1 ...

<?php endif; ?>

...

<?php if (isset($this->blocks['block2'])): ?>

<?= $this->blocks['block2'] ?>

<?php else: ?>

... default content for block2 ...

<?php endif; ?>

...

<?php if (isset($this->blocks['block3'])): ?>

<?= $this->blocks['block3'] ?>

<?php else: ?>

... default content for block3 ...

<?php endif; ?>

...

使用视图组件

[View components](http://www.yiichina.com/doc/api/2.0/yii-base-view)视图组件提供许多视图相关特性， 可创建[yii\base\View](http://www.yiichina.com/doc/api/2.0/yii-base-view)或它的子类实例来获取视图组件， 大多数情况下主要使用 view 应用组件， 可在[应用配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations)中配置该组件，如下所示：

[

// ...

'components' => [

'view' => [

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

],

// ...

],

]

视图组件提供如下实用的视图相关特性，每项详情会在独立章节中介绍：

* [主题](http://www.yiichina.com/doc/guide/2.0/output-theming): 允许为你的Web站点开发和修改主题；
* [片段缓存](http://www.yiichina.com/doc/guide/2.0/caching-fragment): 允许你在Web页面中缓存片段；
* [客户脚本处理](http://www.yiichina.com/doc/guide/2.0/output-client-scripts): 支持CSS 和 JavaScript 注册和渲染；
* [资源包处理](http://www.yiichina.com/doc/guide/2.0/structure-assets): 支持 [资源包](http://www.yiichina.com/doc/guide/2.0/structure-assets)的注册和渲染；
* [模板引擎](http://www.yiichina.com/doc/guide/2.0/tutorial-template-engines): 允许你使用其他模板引擎，如 [Twig](http://twig.sensiolabs.org/), [Smarty](http://www.smarty.net/)。

开发Web页面时，也可能频繁使用以下实用的小特性。

设置页面标题

每个Web页面应有一个标题，正常情况下标题的标签显示在 [布局](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts)中， 但是实际上标题大多由内容视图而不是布局来决定，为解决这个问题， [yii\web\View](http://www.yiichina.com/doc/api/2.0/yii-web-view) 提供 [title](http://www.yiichina.com/doc/api/2.0/yii-web-view#$title-detail) 标题属性可让标题信息从内容视图传递到布局中。

为利用这个特性，在每个内容视图中设置页面标题，如下所示：

<?php

$this->title = 'My page title';

?>

然后在视图中，确保在 <head> 段中有如下代码：

<title><?= Html::encode($this->title) ?></title>

注册Meta元标签

Web页面通常需要生成各种元标签提供给不同的浏览器，如<head>中的页面标题， 元标签通常在布局中生成。

如果想在内容视图中生成元标签，可在内容视图中调用[yii\web\View::registerMetaTag()](http://www.yiichina.com/doc/api/2.0/yii-web-view#registerMetaTag%28%29-detail)方法， 如下所示：

<?php

$this->registerMetaTag(['name' => 'keywords', 'content' => 'yii, framework, php']);

?>

以上代码会在视图组件中注册一个 "keywords" 元标签， 在布局渲染后会渲染该注册的元标签， 然后，如下HTML代码会插入到布局中调用[yii\web\View::head()](http://www.yiichina.com/doc/api/2.0/yii-web-view#head%28%29-detail)方法处：

<meta name="keywords" content="yii, framework, php">

注意如果多次调用 [yii\web\View::registerMetaTag()](http://www.yiichina.com/doc/api/2.0/yii-web-view#registerMetaTag%28%29-detail) 方法， 它会注册多个元标签，注册时不会检查是否重复。

为确保每种元标签只有一个，可在调用方法时指定键作为第二个参数， 例如，如下代码注册两次 "description" 元标签，但是只会渲染第二个。

$this->registerMetaTag(['name' => 'description', 'content' => 'This is my cool website made with Yii!'], 'description');

$this->registerMetaTag(['name' => 'description', 'content' => 'This website is about funny raccoons.'], 'description');

注册链接标签

和 [Meta标签](http://www.yiichina.com/doc/guide/2.0/structure-views#adding-meta-tags) 类似，链接标签有时很实用， 如自定义网站图标，指定Rss订阅，或授权OpenID到其他服务器。 可以和元标签相似的方式调用[yii\web\View::registerLinkTag()](http://www.yiichina.com/doc/api/2.0/yii-web-view#registerLinkTag%28%29-detail)，例如，在内容视图中注册链接标签如下所示：

$this->registerLinkTag([

'title' => 'Live News for Yii',

'rel' => 'alternate',

'type' => 'application/rss+xml',

'href' => 'http://www.yiiframework.com/rss.xml/',

]);

上述代码会转换成

<link title="Live News for Yii" rel="alternate" type="application/rss+xml" href="http://www.yiiframework.com/rss.xml/">

和 [registerMetaTags()](http://www.yiichina.com/doc/api/2.0/yii-web-view#registerMetaTag%28%29-detail) 类似， 调用[registerLinkTag()](http://www.yiichina.com/doc/api/2.0/yii-web-view#registerLinkTag%28%29-detail) 指定键来避免生成重复链接标签。

视图事件

[View components](http://www.yiichina.com/doc/api/2.0/yii-base-view) 视图组件会在视图渲染过程中触发几个事件， 可以在内容发送给终端用户前，响应这些事件来添加内容到视图中或调整渲染结果。

* 该事件可设置 [yii\base\ViewEvent::$isValid](http://www.yiichina.com/doc/api/2.0/yii-base-viewevent#$isValid-detail) 为 false 取消视图渲染。
* [EVENT\_AFTER\_RENDER](http://www.yiichina.com/doc/api/2.0/yii-base-view#EVENT_AFTER_RENDER-detail): 在布局中调用 [yii\base\View::beginPage()](http://www.yiichina.com/doc/api/2.0/yii-base-view#beginPage%28%29-detail) 时触发， 该事件可获取[yii\base\ViewEvent::$output](http://www.yiichina.com/doc/api/2.0/yii-base-viewevent#$output-detail)的渲染结果， 可修改该属性来修改渲染结果。
* [EVENT\_BEGIN\_PAGE](http://www.yiichina.com/doc/api/2.0/yii-base-view#EVENT_BEGIN_PAGE-detail): 在布局调用 [yii\base\View::beginPage()](http://www.yiichina.com/doc/api/2.0/yii-base-view#beginPage%28%29-detail) 时触发；
* [EVENT\_END\_PAGE](http://www.yiichina.com/doc/api/2.0/yii-base-view#EVENT_END_PAGE-detail): 在布局调用 [yii\base\View::endPage()](http://www.yiichina.com/doc/api/2.0/yii-base-view#endPage%28%29-detail) 是触发；
* [EVENT\_BEGIN\_BODY](http://www.yiichina.com/doc/api/2.0/yii-web-view#EVENT_BEGIN_BODY-detail): 在布局调用 [yii\web\View::beginBody()](http://www.yiichina.com/doc/api/2.0/yii-web-view#beginBody%28%29-detail) 时触发；
* [EVENT\_END\_BODY](http://www.yiichina.com/doc/api/2.0/yii-web-view#EVENT_END_BODY-detail): 在布局调用 [yii\web\View::endBody()](http://www.yiichina.com/doc/api/2.0/yii-web-view#endBody%28%29-detail) 时触发。

例如，如下代码将当前日期添加到页面结尾处：

\Yii::$app->view->on(View::EVENT\_END\_BODY, function () {

echo date('Y-m-d');

});

渲染静态页面

静态页面指的是大部分内容为静态的 不需要控制器传递动态数据的Web页面。

可将HTML代码放置在视图中，在控制器中使用以下代码输出静态页面：

public function actionAbout()

{

return $this->render('about');

}

如果Web站点包含很多静态页面，多次重复相似的代码显得很繁琐， 为解决这个问题，可以使用一个在控制器中称为 [yii\web\ViewAction](http://www.yiichina.com/doc/api/2.0/yii-web-viewaction) 的[独立动作](http://www.yiichina.com/doc/guide/2.0/structure-controllers#standalone-actions)。 例如：

namespace app\controllers;

use yii\web\Controller;

class SiteController extends Controller

{

public function actions()

{

return [

'page' => [

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

],

];

}

}

现在如果你在@app/views/site/pages目录下创建名为 about 的视图， 可通过如下rul显示该视图：

http://localhost/index.php?r=site/page&view=about

GET 中 view 参数告知 [yii\web\ViewAction](http://www.yiichina.com/doc/api/2.0/yii-web-viewaction) 动作请求哪个视图，然后操作在 @app/views/site/pages目录下寻找该视图，可配置 [yii\web\ViewAction::$viewPrefix](http://www.yiichina.com/doc/api/2.0/yii-web-viewaction#$viewPrefix-detail) 修改搜索视图的目录。

最佳实践

视图负责将模型的数据展示用户想要的格式，总之，视图

* 应主要包含展示代码，如HTML, 和简单的PHP代码来控制、格式化和渲染数据；
* 不应包含执行数据查询代码，这种代码放在模型中；
* 应避免直接访问请求数据，如 $\_GET, $\_POST，这种应在控制器中执行， 如果需要请求数据，应由控制器推送到视图。
* 可读取模型属性，但不应修改它们。

为使模型更易于维护，避免创建太复杂或包含太多冗余代码的视图， 可遵循以下方法达到这个目标：

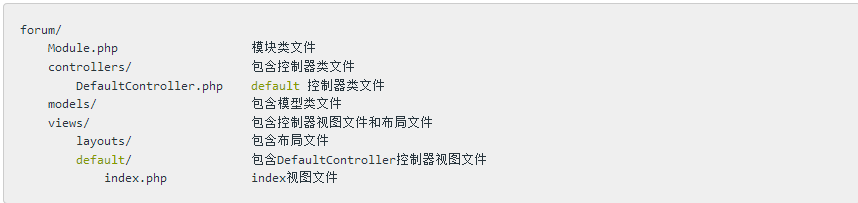
* 使用 [布局](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts) 来展示公共代码（如，页面头部、尾部）；
* 将复杂的视图分成几个小视图， 可使用上面描述的渲染方法将这些小视图渲染并组装成大视图；
* 创建并使用 [小部件](http://www.yiichina.com/doc/guide/2.0/structure-widgets) 作为视图的数据块；
* 创建并使用助手类在视图中转换和格式化数据。

**六、模块**

模块是独立的软件单元，由[模型](http://www.yiichina.com/doc/guide/2.0/structure-models), [视图](http://www.yiichina.com/doc/guide/2.0/structure-views), [控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers)和其他支持组件组成， 终端用户可以访问在[应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications)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](http://www.yiichina.com/doc/guide/2.0/structure-applications)不同的是， 模块不能单独部署，必须属于某个应用主体。

创建模块

模块被组织成一个称为yii\base\Module::basePath的目录， 在该目录中有子目录如controllers, models, views 分别为对应控制器，模型，视图和其他代码，和应用非常类似。 如下例子显示一个模型的目录结构：



模块类

每个模块都有一个继承[yii\base\Module](http://www.yiichina.com/doc/api/2.0/yii-base-module)的模块类， 该类文件直接放在模块的yii\base\Module::basePath目录下， 并且能被 [自动加载](http://www.yiichina.com/doc/guide/2.0/concept-autoloading)。 当一个模块被访问，和 [应用主体实例](http://www.yiichina.com/doc/guide/2.0/structure-applications) 类似会创建该模块类唯一实例，模块实例用来帮模块内代码共享数据和组件。

以下示例一个模块类大致定义：

namespace app\modules\forum;

class Module extends \yii\base\Module

{

public function init()

{

parent::init();

$this->params['foo'] = 'bar';

// ... 其他初始化代码 ...

}

}

如果 init() 方法包含很多初始化模块属性代码， 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

],

];

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。

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。

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，并指向你的命令的命名空间，为了让命令行实用程序看到您的命令，您需要更改 [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';

}

}

然后可以使用以下路由从命令行使用您的命令：

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

使用模块

要在应用中使用模块，只需要将模块加入到应用主体配置的yii\base\Application::modules属性的列表中， 如下代码的[应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations) 使用 forum 模块:

[

'modules' => [

'forum' => [

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

// ... 模块其他配置 ...

],

],

]

yii\base\Application::modules 属性使用模块配置数组， 每个数组键为模块 ID，它标识该应用中唯一的模块， 数组的值为用来创建模块的 [配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations)。

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和动作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 动作， 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) 属性来决定使用哪个控制器/动作， 也就是说路由 forum 可能代表 forum 模块的 default 控制器。

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。

补充: 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。

也可以使用如下方式访问模块实例:

// 获取ID为 "forum" 的模块

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

// 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'];

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%28%29-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',

],

]

模块嵌套

模块可无限级嵌套，也就是说，模块可以包含另一个包含模块的模块， 我们称前者为父模块，后者为子模块， 子模块必须在父模块的yii\base\Module::modules属性中申明，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动作。

**注意:** [getModule()](http://www.yiichina.com/doc/api/2.0/yii-base-module#getModule%28%29-detail) 方法只返回属直接归属于其父级的子模块。 [yii\base\Application::$loadedModules](http://www.yiichina.com/doc/api/2.0/yii-base-application#$loadedModules-detail) 属性保存加载模块的列表， 通过它们的类名索引，包含直接的子节点和嵌套节点。

最佳实践

模块在大型项目中常备使用，这些项目的特性可分组， 每个组包含一些强相关的特性， 每个特性组可以做成一个模块由特定的开发人员和开发组来开发和维护。在特性组上，使用模块也是重用代码的好方式，一些常用特性， 如用户管理，评论管理，可以开发成模块， 这样在相关项目中非常容易被重用。

**七、过滤器**

过滤器是 [控制器 动作](http://www.yiichina.com/doc/guide/2.0/structure-controllers#actions) 执行之前或之后执行的对象。 例如访问控制过滤器可在动作执行之前来控制特殊终端用户是否有权限执行动作， 内容压缩过滤器可在动作执行之后发给终端用户之前压缩响应内容。

过滤器可包含 预过滤（过滤逻辑在动作之前） 或 后过滤（过滤逻辑在动作之后）， 也可同时包含两者。

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',

'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。

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。

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%28%29-detail) 和/或 [afterAction()](http://www.yiichina.com/doc/api/2.0/yii-base-actionfilter#afterAction%28%29-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%28%29-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);

}

}

核心过滤器

Yii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%28%29-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',

],

],

],

];

补充: 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's 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());

}

Cors 可转为使用 cors 属性。

* 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());

}

**八、小部件**

小部件是在 [视图](http://www.yiichina.com/doc/guide/2.0/structure-views) 中使用的可重用单元， 使用面向对象方式创建复杂和可配置用户界面单元。 例如，日期选择器小部件可生成一个精致的允许用户选择日期的日期选择器， 你只需要在视图中插入如下代码：

<?php

use yii\jui\DatePicker;

?>

<?= DatePicker::widget(['name' => 'date']) ?>

Yii提供许多优秀的小部件，比如[active form](http://www.yiichina.com/doc/api/2.0/yii-widgets-activeform), [yii\widgets\Menu|menu]], [jQuery UI widgets](http://www.yiichina.com/doc/guide/2.0/widget-jui), [Twitter Bootstrap widgets](http://www.yiichina.com/doc/guide/2.0/widget-bootstrap)。 接下来介绍小部件的基本知识， 如果你想了解某个小部件请参考对应的类API文档。

使用小部件

小部件基本上在[views](http://www.yiichina.com/doc/guide/2.0/structure-views)中使用，在视图中可调用 [yii\base\Widget::widget()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#widget%28%29-detail) 方法使用小部件。 该方法使用 [配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations) 数组初始化小部件并返回小部件渲染后的结果。 例如如下代码插入一个日期选择器小部件，它配置为使用俄罗斯语， 输入框内容为$model的from\_date属性值。

<?php

use yii\jui\DatePicker;

?>

<?= DatePicker::widget([

'model' => $model,

'attribute' => 'from\_date',

'language' => 'ru',

'clientOptions' => [

'dateFormat' => 'yy-mm-dd',

],

]) ?>

一些小部件可在[yii\base\Widget::begin()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#begin%28%29-detail) 和 [yii\base\Widget::end()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#end%28%29-detail) 调用中使用数据内容。 例如如下代码使用[yii\widgets\ActiveForm](http://www.yiichina.com/doc/api/2.0/yii-widgets-activeform)小部件生成一个登录表单， 小部件会在begin() 和0 end()执行处分别生成<form>的开始标签和结束标签， 中间的任何代码也会被渲染。

<?php

use yii\widgets\ActiveForm;

use yii\helpers\Html;

?>

<?php $form = ActiveForm::begin(['id' => 'login-form']); ?>

<?= $form->field($model, 'username') ?>

<?= $form->field($model, 'password')->passwordInput() ?>

<div class="form-group">

<?= Html::submitButton('Login') ?>

</div>

<?php ActiveForm::end(); ?>

注意和调用 [yii\base\Widget::widget()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#widget%28%29-detail) 返回渲染结果不同， 调用 [yii\base\Widget::begin()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#begin%28%29-detail) 方法返回一个可组建小部件内容的小部件实例。

Configuring global defaults

Global defaults for a widget type could be configured via DI container:

\Yii::$container->set('yii\widgets\LinkPager', ['maxButtonCount' => 5]);

See ["Practical Usage" section in Dependency Injection Container guide](http://www.yiichina.com/doc/guide/2.0/concept-di-container#practical-usage) for details.

创建小部件

继承 [yii\base\Widget](http://www.yiichina.com/doc/api/2.0/yii-base-widget) 类并覆盖 [yii\base\Widget::init()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#init%28%29-detail) 和/或 [yii\base\Widget::run()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#run%28%29-detail) 方法可创建小部件。通常init() 方法处理小部件属性， run() 方法包含小部件生成渲染结果的代码。 渲染结果可在run()方法中直接"echoed"输出或以字符串返回。

如下代码中HelloWidget编码并显示赋给message 属性的值， 如果属性没有被赋值，默认会显示"Hello World"。

namespace app\components;

use yii\base\Widget;

use yii\helpers\Html;

class HelloWidget extends Widget

{

public $message;

public function init()

{

parent::init();

if ($this->message === null) {

$this->message = 'Hello World';

}

}

public function run()

{

return Html::encode($this->message);

}

}

使用这个小部件只需在视图中简单使用如下代码:

<?php

use app\components\HelloWidget;

?>

<?= HelloWidget::widget(['message' => 'Good morning']) ?>

以下是另一种可在begin() 和 end()调用中使用的HelloWidget， HTML编码内容然后显示。

namespace app\components;

use yii\base\Widget;

use yii\helpers\Html;

class HelloWidget extends Widget

{

public function init()

{

parent::init();

ob\_start();

}

public function run()

{

$content = ob\_get\_clean();

return Html::encode($content);

}

}

如上所示，PHP输出缓冲在init()启动，所有在init() 和 run()方法之间的输出内容都会被获取， 并在run()处理和返回。

注意: 当你调用 [yii\base\Widget::begin()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#begin%28%29-detail) 时会创建一个 新的小部件实例并在构造结束时调用init()方法， 在end()时会调用run()方法并输出返回结果。

如下代码显示如何使用这种 HelloWidget:

<?php

use app\components\HelloWidget;

?>

<?php HelloWidget::begin(); ?>

content that may contain <tag>'s

<?php HelloWidget::end(); ?>

有时小部件需要渲染很多内容，一种更好的办法 是将内容放入一个[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)文件， 然后调用[yii\base\Widget::render()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#render%28%29-detail)方法渲染该视图文件，例如：

public function run()

{

return $this->render('hello');

}

小部件的视图文件默认存储在WidgetPath/views目录，WidgetPath代表小部件类文件所在的目录。 假如上述示例小部件类文件在@app/components下， 会渲染@app/components/views/hello.php视图文件。 You may override 可以覆盖[yii\base\Widget::getViewPath()](http://www.yiichina.com/doc/api/2.0/yii-base-widget#getViewPath%28%29-detail)方法自定义视图文件所在路径。

最佳实践

小部件是面向对象方式来重用视图代码。

创建小部件时仍需要遵循MVC模式，通常逻辑代码在小部件类， 展示内容在[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)中。

小部件设计时应是独立的，也就是说使用一个小部件时候， 可以直接丢弃它而不需要额外的处理。 但是当小部件需要外部资源如CSS, JavaScript, 图片等会比较棘手， 幸运的时候Yii提供 [资源包](http://www.yiichina.com/doc/guide/2.0/structure-asset-bundles) 来解决这个问题。

当一个小部件只包含视图代码，它和[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)很相似， 实际上，在这种情况下，唯一的区别是小部件是可以重用类， 视图只是应用中使用的普通PHP脚本。

**九、前端资源**

Yii中的资源是和Web页面相关的文件，可为CSS文件，JavaScript文件，图片或视频等， 资源放在Web可访问的目录下，直接被Web服务器调用。

通过程序自动管理资源更好一点，例如， 当你在页面中使用 yii\jui\DatePicker 小部件时， 它会自动包含需要的CSS和JavaScript文件，而不是要求你手工去找到这些文件并包含， 当你升级小部件时，它会自动使用新版本的资源文件， 在本教程中，我们会详述Yii提供的强大的资源管理功能。

资源包

Yii在资源包中管理资源，资源包简单的说就是放在一个目录下的资源集合， 当在[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)中注册一个资源包， 在渲染Web页面时会包含包中的CSS和JavaScript文件。

**十、扩展**

扩展是专门设计的在 Yii 应用中随时可拿来使用的， 并可重发布的软件包。例如， [yiisoft/yii2-debug](https://github.com/yiisoft/yii2-debug) 扩展在你的应用的每个页面底部添加一个方便用于调试的工具栏， 帮助你简单地抓取页面生成的情况。 你可以使用扩展来加速你的开发过程。

使用扩展

要使用扩展，你要先安装它。大多数扩展以 [Composer](https://getcomposer.org/) 软件包的形式发布， 这样的扩展可采取下述两个步骤来安装：

1. 修改你的应用的 composer.json 文件，指明你要安装的是哪个扩展 （Composer 软件包）。
2. 运行 composer install 来安装指定的扩展。

注意如果你还没有安装 [Composer](https://getcomposer.org/) ，你需要先安装。

默认情况，Composer安装的是在 [Packagist](https://packagist.org/)中注册的软件包 - 最大的开源 Composer 代码库。你可以在 Packageist 中查找扩展。 你也可以 [创建你自己的代码库](https://getcomposer.org/doc/05-repositories.md#repository) 然后配置 Composer 来使用它。 如果是在开发私有的扩展，并且想只在你的其他工程中共享时，这样做是很有用的。

通过 Composer 安装的扩展会存放在 BasePath/vendor 目录下，这里的 BasePath 指你的应用的 [base path](http://www.yiichina.com/doc/guide/2.0/structure-applications#basePath)。因为 Composer 还是一个依赖管理器，当它安装一个包时， 也将安装这个包所依赖的所有软件包。

例如想安装 yiisoft/yii2-imagine 扩展，可按如下示例修改你的 composer.json 文件：

{

// ...

"require": {

// ... other dependencies

"yiisoft/yii2-imagine": "~2.0.0"

}

}

安装完成后，你应该能在 BasePath/vendor 目录下见到 yiisoft/yii2-imagine 目录。你也应该 见到另一个 imagine/imagine 目录，在其中安装了所依赖的包。

yiisoft/yii2-imagine 是 Yii 由开发团队维护一个核心扩展， 所有核心扩展均由 [Packagist](https://packagist.org/) 集中管理，命名为 yiisoft/yii2-xyz，其中的 xyz， 不同扩展有不同名称。

现在你可以使用安装好的扩展了，好比是应用的一部分。如下示例展示了如何使用 yiisoft/yii2-imagine 扩展 提供的 yii\imagine\Image 类：

use Yii;

use yii\imagine\Image;

// generate a thumbnail image

Image::thumbnail('@webroot/img/test-image.jpg', 120, 120)

->save(Yii::getAlias('@runtime/thumb-test-image.jpg'), ['quality' => 50]);

**注:** 扩展类由 [Yii class autoloader](http://www.yiichina.com/doc/guide/2.0/concept-autoloading) 自动加载。

手动安装扩展

创建扩展

核心扩展

Yii 提供了下列核心扩展，由 Yii 开发团队开发和维护。这些扩展全都在 [Packagist](https://packagist.org/) 中注册，并像 [Using Extensions](http://www.yiichina.com/doc/guide/2.0/structure-extensions#using-extensions) 章节描述 的那样容易安装。

* [yiisoft/yii2-apidoc](https://github.com/yiisoft/yii2-apidoc): 提供了一个可扩展的、高效的 API 文档生成器。核心框架的 API 文档也是用它生成的。
* [yiisoft/yii2-authclient](https://github.com/yiisoft/yii2-authclient): 提供了一套常用的认证客户端，例如 Facebook OAuth2 客户端、GitHub OAuth2 客户端。
* [yiisoft/yii2-bootstrap](https://github.com/yiisoft/yii2-bootstrap): 提供了一套挂件，封装了 [Bootstrap](http://getbootstrap.com/) 的组件和插件。
* [yiisoft/yii2-codeception](https://github.com/yiisoft/yii2-codeception): 提供了基于 [Codeception](http://codeception.com/) 的测试支持。
* [yiisoft/yii2-debug](https://github.com/yiisoft/yii2-debug): 提供了对 Yii 应用的调试支持。当使用该扩展是， 在每个页面的底部将显示一个调试工具条。 该扩展还提供了一个独立的页面，以显示更详细的调试信息。
* [yiisoft/yii2-elasticsearch](https://github.com/yiisoft/yii2-elasticsearch): 提供对 [Elasticsearch](http://www.elasticsearch.org/) 的使用支持。它包含基本的查询/搜索支持， 并实现了 [Active Record](http://www.yiichina.com/doc/guide/2.0/db-active-record) 模式让你可以将活动记录 存储在 Elasticsearch 中。
* [yiisoft/yii2-faker](https://github.com/yiisoft/yii2-faker): 提供了使用 [Faker](https://github.com/fzaninotto/Faker) 的支持，为你生成模拟数据。
* [yiisoft/yii2-gii](https://github.com/yiisoft/yii2-gii): 提供了一个基于页面的代码生成器，具有高可扩展性，并能用来快速生成模型、 表单、模块、CRUD等。
* [yiisoft/yii2-httpclient](https://github.com/yiisoft/yii2-httpclient): provides an HTTP client.
* [yiisoft/yii2-imagine](https://github.com/yiisoft/yii2-imagine): 提供了基于 [Imagine](http://imagine.readthedocs.org/) 的常用图像处理功能。
* [yiisoft/yii2-jui](https://github.com/yiisoft/yii2-jui): 提供了一套封装 [JQuery UI](http://jqueryui.com/) 的挂件以及它们的交互。
* [yiisoft/yii2-mongodb](https://github.com/yiisoft/yii2-mongodb): 提供了对 [MongoDB](http://www.mongodb.org/) 的使用支持。它包含基本 的查询、活动记录、数据迁移、缓存、代码生成等特性。
* [yiisoft/yii2-redis](https://github.com/yiisoft/yii2-redis): 提供了对 [redis](http://redis.io/) 的使用支持。它包含基本的 查询、活动记录、缓存等特性。
* [yiisoft/yii2-smarty](https://github.com/yiisoft/yii2-smarty): 提供了一个基于 [Smarty](http://www.smarty.net/) 的模板引擎。
* [yiisoft/yii2-sphinx](https://github.com/yiisoft/yii2-sphinx): 提供了对 [Sphinx](http://sphinxsearch.com) 的使用支持。它包含基本的 查询、活动记录、代码生成等特性。
* [yiisoft/yii2-swiftmailer](https://github.com/yiisoft/yii2-swiftmailer): 提供了基于 [swiftmailer](http://swiftmailer.org/) 的邮件发送功能。
* [yiisoft/yii2-twig](https://github.com/yiisoft/yii2-twig): 提供了一个基于 [Twig](http://twig.sensiolabs.org/) 的模板引擎。