一、总览

Yii 应用参照[模型-视图-控制器 （MVC）](http://wikipedia.org/wiki/Model-view-controller) 设计模式来组织。

[模型](http://www.yiichina.com/doc/guide/2.0/structure-models)代表数据、业务逻辑和规则；

[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)展示模型的输出；

[控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 接受出入并将其转换为[模型](http://www.yiichina.com/doc/guide/2.0/structure-models)和[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)命令。

除了 MVC, Yii 应用还有以下部分：

* [入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)：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](http://www.yiichina.com/doc/guide/2.0/structure-applications)：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](http://www.yiichina.com/doc/guide/2.0/structure-application-components)：在应用中注册的对象， 提供不同的功能来完成请求。
* [模块](http://www.yiichina.com/doc/guide/2.0/structure-modules)：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](http://www.yiichina.com/doc/guide/2.0/structure-filters)：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](http://www.yiichina.com/doc/guide/2.0/structure-widgets)：可嵌入到[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)中的对象， 可包含控制器逻辑，可被不同视图重复调用。

二、入口脚本

入口脚本是应用启动流程中的第一环， 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()-detail) 来处理请求。

## Web 应用

以下是[基础应用模版](http://www.yiichina.com/doc/guide/2.0/start-installation)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，注意这是一个应用

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

（控制台应用，暂不考虑）

## 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，详情请查阅 [配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations#environment-constants)章节。 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**);

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。

三、应用

1. [应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations)
2. [应用主体属性](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-properties)
3. [应用事件](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-events)
4. [应用主体生命周期](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-lifecycle)

## 应用主体

应用主体是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](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) 中创建并能通过表达式 \Yii::$app 全局范围内访问。

Yii有两种应用主体: [网页应用主体](http://www.yiichina.com/doc/api/2.0/yii-web-application) and [控制台应用主体](http://www.yiichina.com/doc/api/2.0/yii-console-application)， 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](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) 创建了一个应用主体， 它会加载一个 [配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations) 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();

类似其他 [配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations) 文件, 应用主体配置文件标明如何设置应用对象初始属性。

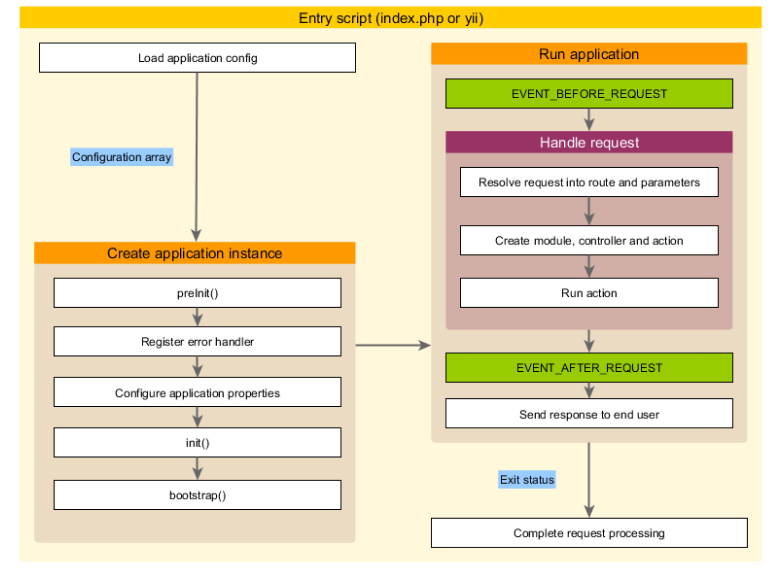
由于应用主体配置比较复杂，一般保存在多个类似如上web.php的[配置文件](http://www.yiichina.com/doc/guide/2.0/concept-configurations#configuration-files) 当中。

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](http://www.yiichina.com/doc/guide/2.0/structure-controllers) ， 临时文件保存到哪儿等等。 以下我们简述这些属性。

应用事件：

## 应用主体生命周期



当运行 [入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) 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()](http://www.yiichina.com/doc/api/2.0/yii-base-application#preInit()-detail) 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()](http://www.yiichina.com/doc/api/2.0/yii-base-application#init()-detail) 初始化，该函数会调用 [bootstrap()](http://www.yiichina.com/doc/api/2.0/yii-base-application#bootstrap()-detail) 运行引导启动组件.
3. 入口脚本调用 [yii\base\Application::run()](http://www.yiichina.com/doc/api/2.0/yii-base-application#run()-detail) 运行应用主体:
   * 触发 [EVENT\_BEFORE\_REQUEST](http://www.yiichina.com/doc/api/2.0/yii-base-application#EVENT_BEFORE_REQUEST-detail) 事件。
   * 处理请求：解析请求 [路由](http://www.yiichina.com/doc/guide/2.0/runtime-routing) 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](http://www.yiichina.com/doc/api/2.0/yii-base-application#EVENT_AFTER_REQUEST-detail) 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。

（应用主体是如何使用的，现在还不明白）

四、应用组件

1. [引导启动组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components#bootstrapping-components)
2. [核心应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components#core-application-components)

应用主体是[服务定位器](http://www.yiichina.com/doc/guide/2.0/concept-service-locator)， 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

如：

可以使用 \Yii::$app->db 来获取到已注册到应用的 [DB connection](http://www.yiichina.com/doc/api/2.0/yii-db-connection)，

使用 \Yii::$app->cache 来获取到已注册到应用的 [primary cache](http://www.yiichina.com/doc/api/2.0/yii-caching-cache)。

应用组件可以是任意对象，可以在 [应用主体配置](http://www.yiichina.com/doc/guide/2.0/structure-applications#application-configurations)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;

},

],

]

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

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](http://www.yiichina.com/doc/api/2.0/yii-base-application#bootstrap()-detail) 属性中。

还可以使用闭包来引导启动自定义的组件。不需要直接返回一个实例化的组件。 在应用主体 [yii\base\Application](http://www.yiichina.com/doc/api/2.0/yii-base-application) 实例化后，闭包也会被调用。

例如, 如下的应用主体配置保证了 log 组件一直被加载。

[

'bootstrap' => [

'log',

**function**($app){

**return** **new** ComponentX();

},

**function**($app){

// 可以写自定义的代码

**return**;

}

],

'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](http://www.yiichina.com/doc/guide/2.0/runtime-routing)；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。

下面是预定义的核心应用组件列表， 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。

* 详情请参考 [管理资源](http://www.yiichina.com/doc/guide/2.0/output-assets) 一节。
* 注意配置该组件时必须指定组件类名和其他相关组件属性， 如[yii\db\Connection::$dsn](http://www.yiichina.com/doc/api/2.0/yii-db-connection#$dsn-detail)。 详情请参考 [数据访问对象](http://www.yiichina.com/doc/guide/2.0/db-dao) 一节。
* yii\base\Application::errorHandler: 处理 PHP 错误和异常， 详情请参考 [错误处理](http://www.yiichina.com/doc/guide/2.0/tutorial-handling-errors) 一节。
* 日期使用长格式。 详情请参考 [格式化输出数据](http://www.yiichina.com/doc/guide/2.0/output-formatting) 一节。
* 详情请参考 [国际化](http://www.yiichina.com/doc/guide/2.0/tutorial-i18n) 一节。
* 详情请参考 [日志](http://www.yiichina.com/doc/guide/2.0/tutorial-logging) 一节。
* 详情请参考 [邮件](http://www.yiichina.com/doc/guide/2.0/tutorial-mailing) 一节。
* 详情请参考 [响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses) 一节。
* 详情请参考 [请求](http://www.yiichina.com/doc/guide/2.0/runtime-requests) 一节。
* 仅在[Web applications](http://www.yiichina.com/doc/api/2.0/yii-web-application) 网页应用中可用， 详情请参考 [Sessions (会话) and Cookies](http://www.yiichina.com/doc/guide/2.0/runtime-sessions-cookies) 一节。
* 详情请参考 [URL 解析和生成](http://www.yiichina.com/doc/guide/2.0/runtime-url-handling) 一节。
* 仅在[Web applications](http://www.yiichina.com/doc/api/2.0/yii-web-application) 网页应用中可用， 详情请参考 [认证](http://www.yiichina.com/doc/guide/2.0/security-authentication) 一节。
* 详情请参考 [Views](http://www.yiichina.com/doc/guide/2.0/structure-views) 一节。

五、控制器

1. [动作](http://www.yiichina.com/doc/guide/2.0/structure-controllers#actions)
2. [路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes)
3. [创建控制器](http://www.yiichina.com/doc/guide/2.0/structure-controllers#creating-controllers)
4. [创建动作](http://www.yiichina.com/doc/guide/2.0/structure-controllers#creating-actions)
5. [控制器生命周期](http://www.yiichina.com/doc/guide/2.0/structure-controllers#controller-lifecycle)
6. [最佳实践](http://www.yiichina.com/doc/guide/2.0/structure-controllers#best-practices)

控制器是 [MVC](http://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller) 模式中的一部分， 是继承[yii\base\Controller](http://www.yiichina.com/doc/api/2.0/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](http://www.yiichina.com/doc/guide/2.0/structure-applications) 接管控制后会分析请求数据并传送到[模型](http://www.yiichina.com/doc/guide/2.0/structure-models)， 传送模型结果到[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)，最后生成输出响应信息。

## 动作

控制器由 操作 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操作。

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,

]);

}

}

}

在操作 view (定义为 actionView() 方法)中， 代码首先根据请求模型ID加载 [模型](http://www.yiichina.com/doc/guide/2.0/structure-models)， 如果加载成功，会渲染名称为view的[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)并显示，否则会抛出一个异常。

在操作 create (定义为 actionCreate() 方法)中, 代码相似. 先将请求数据填入[模型](http://www.yiichina.com/doc/guide/2.0/structure-models)， 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操作， 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：

* 模块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;
* 操作ID: 同控制器下唯一标识操作的字符串。

路由使用如下格

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 操作

## 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 是真是的控制器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](http://www.yiichina.com/doc/api/2.0/yii-base-application#$controllerNamespace-detail) 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。

## 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（当不能对应时这样强制配置）。

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

创建操作就是在控制器类中定义 操作方法 来完成，操作方法必须是以action开头的公有方法。 操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操作 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';

}

}

## 操作ID

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

操作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是真实的操作ID， view?, Update不是操作ID.

可通过两种方式创建操作ID，内联操作和独立操作. An inline action is 内联操作在控制器类中定义为方法；独立操作是继承[yii\base\Action](http://www.yiichina.com/doc/api/2.0/yii-base-action)或它的子类的类。 内联操作容易创建，在无需重用的情况下优先使用； 独立操作相反，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)。

## 内联操作（就是我们自己在控制器中定义的操作）

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：

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

例如index 转成 actionIndex, hello-world 转成 actionHelloWorld。

操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操作会返回一个异常， 也要注意操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操作。

## 独立动作

独立操作通过继承[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)都是独立操作。

要使用独立操作，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()-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() 方法返回键为操作ID、值为对应操作类名 或数组[configurations](http://www.yiichina.com/doc/guide/2.0/concept-configurations) 的数组。 和内联操作不同，独立操作ID可包含任意字符，只要在actions() 方法中申明.

为创建一个独立操作类，需要继承[yii\base\Action](http://www.yiichina.com/doc/api/2.0/yii-base-action) 或它的子类，并实现公有的名称为run()的方法, run() 方法的角色和操作方法类似，例如：

<?php

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

**use** **yii**\**base**\**Action**;

**class** **HelloWorldAction** **extends** **Action**

{

**public** **function** **run**()

{

**return** "Hello World";

}

}

## 动作结果

操作方法或独立操作的run()方法的返回值非常重要， 它表示对应操作结果。

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，操作结果都为字符串，作为响应数据发送给终端用户， 下例显示一个操作通过 返回响应对象（因为[redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-controller#redirect()-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。

## 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) 属性指定的默认操作， 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操作。

默认操作默认为 index，如果想修改默认操作，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：

1. 在控制器创建和配置后，yii\base\Controller::init() 方法会被调用。
2. 控制器根据请求操作ID创建一个操作对象:
   * 如果操作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()-detail)找到操作ID， 会创建一个独立操作；
   * 如果操作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。
3. 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()会跳过并且操作执行会被取消； 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操作；
4. 控制器执行操作:
   * 请求数据解析和填入到操作参数；
5. 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操作；
6. 应用主体获取操作结果并赋值给[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，包含的操作代码简短； 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.

六、模型

1. [属性](http://www.yiichina.com/doc/guide/2.0/structure-models#attributes)
2. [场景](http://www.yiichina.com/doc/guide/2.0/structure-models#scenarios)
3. [验证规则](http://www.yiichina.com/doc/guide/2.0/structure-models#validation-rules)
4. [块赋值](http://www.yiichina.com/doc/guide/2.0/structure-models#massive-assignment)
5. [数据导出](http://www.yiichina.com/doc/guide/2.0/structure-models#data-exporting)
6. [最佳实践](http://www.yiichina.com/doc/guide/2.0/structure-models#best-practices)

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。

## 属性

模型通过 属性 来代表业务数据，每个属性像是模型的公有可访问属性， [yii\base\Model::attributes()](http://www.yiichina.com/doc/api/2.0/yii-base-model#attributes()-detail) 指定模型所拥有的属性。可像访问一个对象属性一样访问模型的属性:

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

// "name" 是ContactForm模型的属性

$model->name = 'example';

**echo** $model->name;

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

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

// 像访问数组单元项一样访问属性

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

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

// 迭代器遍历模型

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

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

}

## 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()-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()-detail) 获取属性的标签，例如：

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

// 显示为 "Name"

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

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

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

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

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

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

{

**public** $name;

**public** $email;

**public** $subject;

**public** $body;

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

{

**return** [

'name' => 'Your name',

'email' => 'Your email address',

'subject' => 'Subject',

'body' => 'Content',

];

}

}

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

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

{

**return** [

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

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

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

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

];

}

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

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

## 场景

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

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

// 场景作为属性来设置

$model = **new** User;

$model->scenario = 'login';

// 场景通过构造初始化配置来设置

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

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

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

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

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

{

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

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

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

{

**return** [

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

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

];

}

}

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

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

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

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

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

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

{

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

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

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

{

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

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

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

**return** $scenarios;

}

}

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

## 验证规则

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

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

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

// 用户输入数据赋值到模型属性

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

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

// 所有输入数据都有效 all inputs are valid

} **else** {

// 验证失败：$errors 是一个包含错误信息的数组

$errors = $model->errors;

}

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

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

{

**return** [

// name, email, subject 和 body 属性必须有值

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

// email 属性必须是一个有效的电子邮箱地址

['email', 'email'],

];

}

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

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

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

{

**return** [

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

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

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

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

];

}

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

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

## 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()-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()-detail)方法 列出的称之为 安全属性 的属性上，例如，如果User模型申明以下场景， 当当前场景为login时候，只有username and password 可被块赋值， 其他属性不会被赋值。

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

{

**return** [

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

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

];

}

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

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

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

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

{

**return** [

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

];

}

## 非安全属性

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

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

{

**return** [

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

];

}

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

$model->secret = $secret;

The same can be done in rules() method:

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

{

**return** [

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

];

}

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

## 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：

* 模型转换成数组；
* 数组转换成所需要的格式。

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

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

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

$array = $post->attributes;

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

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

## 字段

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

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

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

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

// 明确列出每个字段，特别用于你想确保数据表或模型

// 属性改变不会导致你的字段改变(保证后端的API兼容)。

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

{

**return** [

// 字段名和属性名相同

'id',

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

'email' => 'email\_address',

// 字段名为 "name", 值通过PHP代码返回

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

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

},

];

}

// 过滤掉一些字段，特别用于

// 你想继承父类实现并不想用一些敏感字段

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

{

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

// 去掉一些包含敏感信息的字段

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

**return** $fields;

}

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

## 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。

视图

1. [创建视图](http://www.yiichina.com/doc/guide/2.0/structure-views#creating-views)
2. [渲染视图](http://www.yiichina.com/doc/guide/2.0/structure-views#rendering-views)
3. [布局](http://www.yiichina.com/doc/guide/2.0/structure-views#layouts)
4. [使用视图组件](http://www.yiichina.com/doc/guide/2.0/structure-views#using-view-components)
5. [视图事件](http://www.yiichina.com/doc/guide/2.0/structure-views#view-events)
6. [渲染静态页面](http://www.yiichina.com/doc/guide/2.0/structure-views#rendering-static-pages)
7. [最佳实践](http://www.yiichina.com/doc/guide/2.0/structure-views#best-practices)

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。

**提示：**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()-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()-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()-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()-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()-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()-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()-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()-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()-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。

## 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()-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()-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。

**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 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。