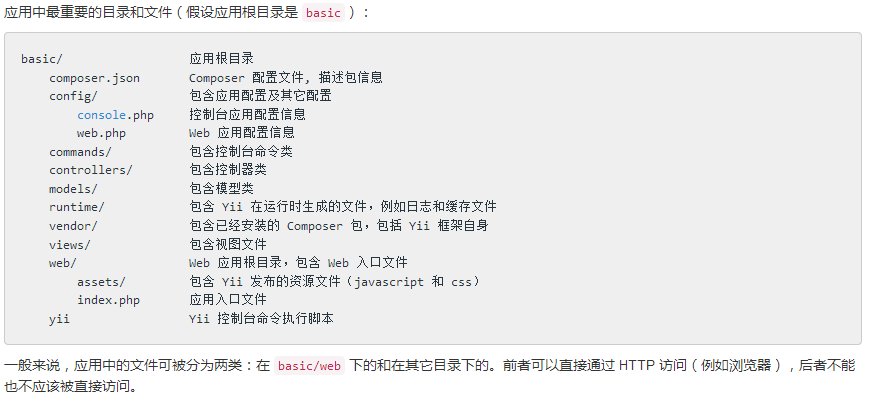
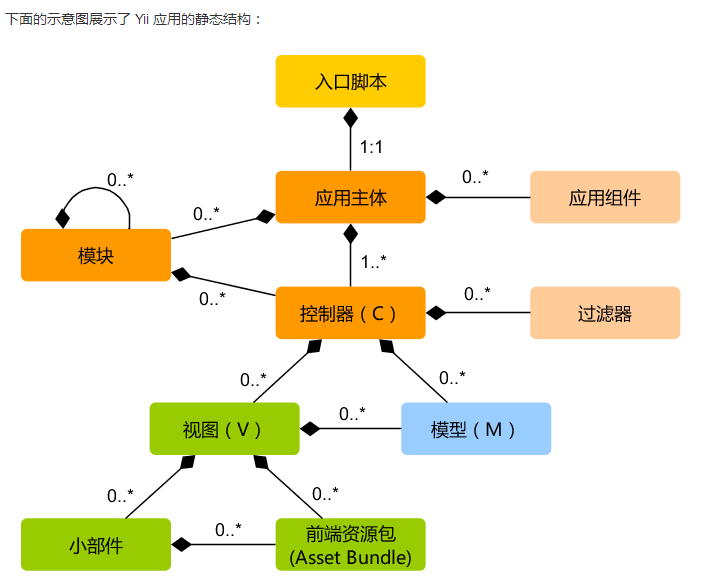
**零、Yii框架的应用结构**



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.

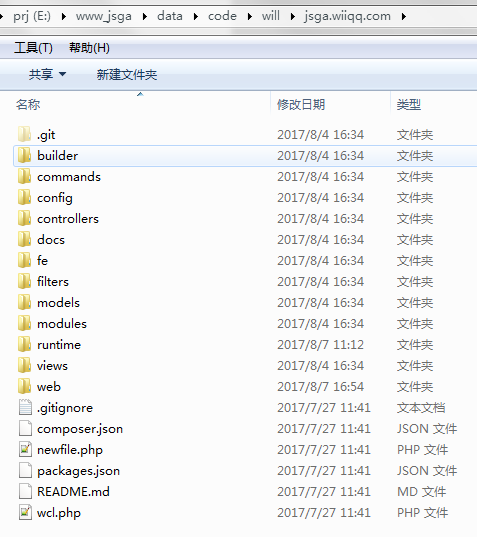
应用组件：在应用中注册的对象， 提供不同的功能来完成请求。

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。

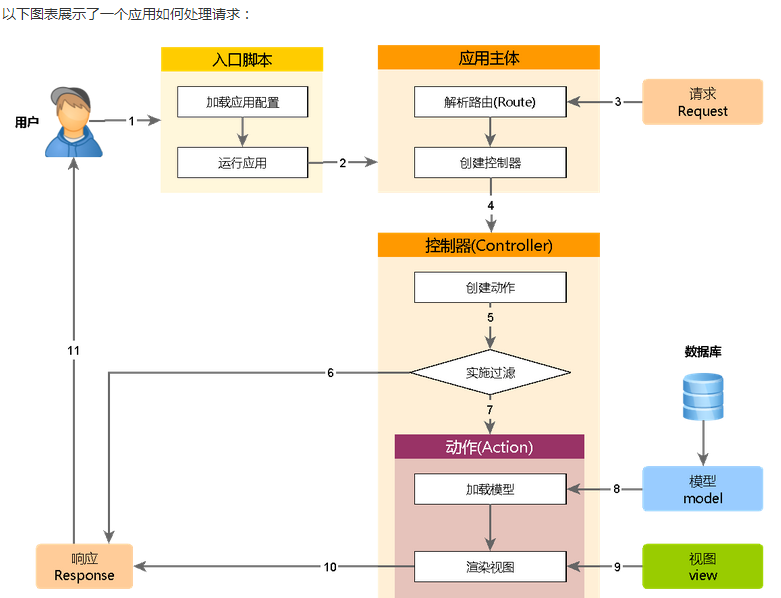
小部件：可嵌入到视图中的对象， 可包含控制器逻辑，可被不同视图重复调用。

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。

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

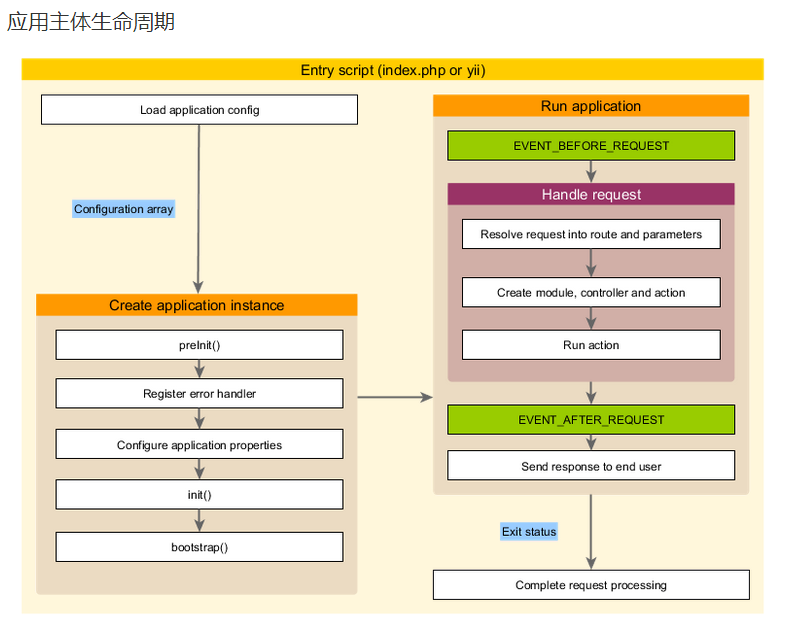
} else {

}

},

]

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" \l "$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" \l "$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" \l "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::$data](http://www.yiichina.com/doc/api/2.0/yii-web-response" \l "$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" \l "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" \l "$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.