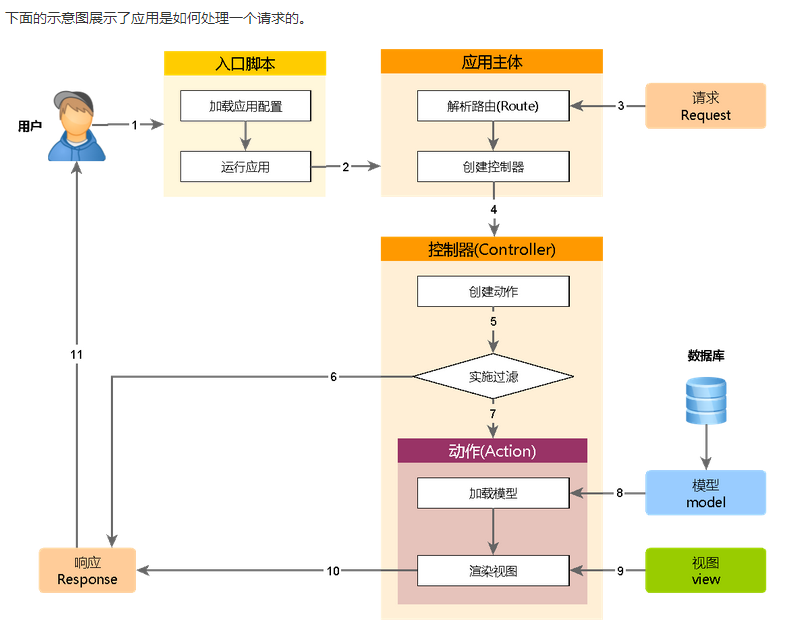
**零、运行机制概述**

每一次 Yii 应用开始处理 HTTP 请求时，它都会进行一个近似的流程。

1. 用户提交指向 [入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) web/index.php 的请求。
2. 入口脚本会加载 [配置数组](http://www.yiichina.com/doc/guide/2.0/concept-configurations) 并创建一个 [应用](http://www.yiichina.com/doc/guide/2.0/structure-applications) 实例用于处理该请求。
3. 应用会通过 [request（请求）](http://www.yiichina.com/doc/guide/2.0/runtime-requests) 应用组件解析被请求的 [路由](http://www.yiichina.com/doc/guide/2.0/runtime-routing)。
4. 应用创建一个 [controller（控制器）](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 实例具体处理请求。
5. 控制器会创建一个 [action（动作）](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 实例并为该动作执行相关的 Filters（访问过滤器）。
6. 如果任何一个过滤器验证失败，该动作会被取消。
7. 如果全部的过滤器都通过，该动作就会被执行。
8. 动作会加载一个数据模型，一般是从数据库中加载。
9. 动作会渲染一个 View（视图），并为其提供所需的数据模型。
10. 渲染得到的结果会返回给 [response（响应）](http://www.yiichina.com/doc/guide/2.0/runtime-responses) 应用组件。
11. 响应组件会把渲染结果发回给用户的浏览器。



**一、启动引导（Bootstrapping）**

启动引导是指：在应用开始解析并处理新接受请求之前，一个预先准备环境的过程。 启动引导会在两个地方具体进行：[入口脚本(Entry Script)](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts) 和 [应用主体（application）](http://www.yiichina.com/doc/guide/2.0/structure-applications)。

在[入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)里，需注册各个类库的类文件自动加载器（Class Autoloader，简称自动加载器）。 这主要包括通过其 autoload.php 文件加载的Composer 自动加载器，以及通过 Yii 类加载的 Yii 自动加载器。之后， 入口脚本会加载应用的[配置（configuration）](http://www.yiichina.com/doc/guide/2.0/concept-configurations)并创建一个 [应用主体](http://www.yiichina.com/doc/guide/2.0/structure-applications) 的实例。

在应用主体的构造函数中，会执行以下引导工作：

1. 调用 [preInit()](http://www.yiichina.com/doc/api/2.0/yii-base-application#preInit%28%29-detail)（预初始化）方法，配置一些高优先级的应用属性， 比如 yii\base\Application::basePath 属性。
2. 注册yii\base\Application::errorHandler。
3. 通过给定的应用配置初始化应用的各属性。
4. 通过调用 [init()](http://www.yiichina.com/doc/api/2.0/yii-base-application#init%28%29-detail)（初始化）方法，它会顺次调用 [bootstrap()](http://www.yiichina.com/doc/api/2.0/yii-base-application#bootstrap%28%29-detail) 从而运行引导组件。
5. 加载扩展清单文件(extension manifest file) vendor/yiisoft/extensions.php。
6. 创建并运行各个扩展声明的 [引导组件（bootstrap components）](http://www.yiichina.com/doc/guide/2.0/structure-extensions#bootstrapping-classes)。
7. 创建并运行各个 [应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components) 以及在应用的 [Bootstrap 属性](http://www.yiichina.com/doc/guide/2.0/structure-applications#bootstrap)中声明的各个 [模块（modules）组件](http://www.yiichina.com/doc/guide/2.0/structure-modules)（如果有）。

因为引导工作必须在处理**每一次**请求之前都进行一遍， 因此让该过程尽可能轻量化就异常重要，请尽可能地优化这一步骤。

请尽量不要注册太多引导组件。只有他需要在 HTTP 请求处理的全部生命周期中都作用时才需要使用它。 举一个用到它的范例：一个模块需要注册额外的 URL 解析规则，就应该把它列在应用的 [bootstrap 属性](http://www.yiichina.com/doc/guide/2.0/structure-applications#bootstrap)之中， 这样该 URL 解析规则才能在解析请求之前生效。 （译注：换言之，为了性能需要，除了 URL 解析等少量操作之外，绝大多数组件都应该按需加载，而不是都放在引导过程中。）

在生产环境中，可以开启字节码缓存，比如 APC， 来进一步最小化加载和解析 PHP 文件所需的时间。

一些大型应用都包含有非常复杂的应用[配置](http://www.yiichina.com/doc/guide/2.0/concept-configurations)， 它们会被分割到许多更小的配置文件中。 此时，可以考虑将整个配置数组缓存起来， 并在入口脚本创建应用实例之前直接从缓存中加载。

**二、路由**

当[入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)在调用 yii\web\Application::run() 方法时，它进行的第一个操作就是解析输入的请求，然后实例化对应的[控制器操作](http://www.yiichina.com/doc/guide/2.0/structure-controllers)处理这个请求。 该过程就被称为**引导路由（routing）**。

The central piece responsible for routing and URL creation is the [URL manager](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager), which is registered as the urlManager [application component](http://www.yiichina.com/doc/guide/2.0/structure-application-components). The [URL manager](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager) provides the [parseRequest()](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#parseRequest%28%29-detail) method to parse an incoming request into a route and the associated query parameters and the [createUrl()](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#createUrl%28%29-detail) method to create a URL from a given route and its associated query parameters.

By configuring the urlManager component in the application configuration, you can let your application recognize arbitrary URL formats without modifying your existing application code. For example, you can use the following code to create a URL for the post/view action:

use yii\helpers\Url;

// Url::to() calls UrlManager::createUrl() to create a URL

$url = Url::to(['post/view', 'id' => 100]);

Depending on the urlManager configuration, the created URL may look like one of the following (or other format). And if the created URL is requested later, it will still be parsed back into the original route and query parameter value.

/index.php?r=post%2Fview&id=100

/index.php/post/100

/posts/100

URL Formats

The [URL manager](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager) supports two URL formats: the default URL format and the pretty URL format.

The default URL format uses a query parameter named r to represent the route and normal query parameters to represent the query parameters associated with the route. For example, the URL /index.php?r=post/view&id=100 represents the route post/view and the id query parameter 100.

The default URL format does not require any configuration of the [URL manager](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager) and works in any Web server setup.

The pretty URL format uses the extra path following the entry script name to represent the route and the associated query parameters. For example, the extra path in the URL /index.php/post/100 is /post/100 which may represent the route post/view and the id query parameter 100 with a proper [URL rule](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#$rules-detail). To use the pretty URL format, you will need to design a set of [URL rules](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#$rules-detail) according to the actual requirement about how the URLs should look like.

You may switch between the two URL formats by toggling the [enablePrettyUrl](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#$enablePrettyUrl-detail) property of the [URL manager](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager) without changing any other application code.

Routing

Routing involves two steps. In the first step, the incoming request is parsed into a route and the associated query parameters. In the second step, a [controller action](http://www.yiichina.com/doc/guide/2.0/structure-controllers#actions) corresponding to the parsed route is created to handle the request.

When using the default URL format, parsing a request into a route is as simple as getting the value of a GET query parameter named r.

When using the pretty URL format, the [URL manager](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager) will examine the registered [URL rules](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#$rules-detail) to find matching one that can resolve the request into a route. If such a rule cannot be found, a [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception) exception will be thrown.

Once the request is parsed into a route, it is time to create the controller action identified by the route. The route is broken down into multiple parts by the slashes in it. For example, site/index will be broken into site and index. Each part is an ID which may refer to a module, a controller or an action. Starting from the first part in the route, the application takes the following steps to create modules (if any), controller and action:

1. Set the application as the current module.
2. Check if the [controller map](http://www.yiichina.com/doc/api/2.0/yii-base-module#$controllerMap-detail) of the current module contains the current ID. If so, a controller object will be created according to the controller configuration found in the map, and Step 5 will be taken to handle the rest part of the route.
3. Check if the ID refers to a module listed in the yii\base\Module::modules property of the current module. If so, a module is created according to the configuration found in the module list, and Step 2 will be taken to handle the next part of the route under the context of the newly created module.
4. Treat the ID as a [controller ID](http://www.yiichina.com/doc/guide/2.0/structure-controllers#controller-ids) and create a controller object. Do the next step with the rest part of the route.
5. The controller looks for the current ID in its [action map](http://www.yiichina.com/doc/api/2.0/yii-base-controller#actions%28%29-detail). If found, it creates an action according to the configuration found in the map. Otherwise, the controller will attempt to create an inline action which is defined by an action method corresponding to the current [action ID](http://www.yiichina.com/doc/guide/2.0/structure-controllers#action-ids).

Among the above steps, if any error occurs, a [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception) will be thrown, indicating the failure of the routing process.

缺省路由

如果传入请求并没有提供一个具体的路由，（一般这种情况多为于对首页的请求）此时就会启用由 [yii\web\Application::$defaultRoute](http://www.yiichina.com/doc/api/2.0/yii-web-application#$defaultRoute-detail) 属性所指定的缺省路由。 该属性的默认值为 site/index，它指向 site 控制器的 index 动作。你可以像这样在应用配置中调整该属性的值：

return [

// ...

'defaultRoute' => 'main/index',

];

catchAll 路由（全拦截路由）

有时候，你会想要将你的 Web 应用临时调整到维护模式，所有的请求下都会显示相同的信息页。当然，要实现这一点有很多种方法。这里面最简单快捷的方法就是在应用配置中设置下 [yii\web\Application::$catchAll](http://www.yiichina.com/doc/api/2.0/yii-web-application#$catchAll-detail) 属性：

return [

// ...

'catchAll' => ['site/offline'],

];

With the above configuration, the site/offline action will be used to handle all incoming requests.

The catchAll property should take an array whose first element specifies a route, and the rest of the elements (name-value pairs) specify the parameters to be [bound to the action](http://www.yiichina.com/doc/guide/2.0/structure-controllers#action-parameters).

**Info:** Debug panel on development environment will not work when this property is enabled

Creating URLs

Yii provides a helper method yii\helpers\Url::to() to create various kinds of URLs from given routes and their associated query parameters. For example,

use yii\helpers\Url;

// creates a URL to a route: /index.php?r=post%2Findex

echo Url::to(['post/index']);

// creates a URL to a route with parameters: /index.php?r=post%2Fview&id=100

echo Url::to(['post/view', 'id' => 100]);

// creates an anchored URL: /index.php?r=post%2Fview&id=100#content

echo Url::to(['post/view', 'id' => 100, '#' => 'content']);

// creates an absolute URL:http://www.example.com/index.php?r=post%2Findex

echo Url::to(['post/index'], true);

// creates an absolute URL using the https scheme: https://www.example.com/index.php?r=post%2Findex

echo Url::to(['post/index'], 'https');

Note that in the above example, we assume the default URL format is being used. If the pretty URL format is enabled, the created URLs will be different, according to the [URL rules](http://www.yiichina.com/doc/api/2.0/yii-web-urlmanager#$rules-detail) in use.

Starting from version 2.0.2, you may specify a route in terms of an [alias](http://www.yiichina.com/doc/guide/2.0/concept-aliases). If this is the case, the alias will first be converted into the actual route which will then be turned into an absolute route according to the above rules.

For example, assume the current module is admin and the current controller is post,

use yii\helpers\Url;

// currently requested route: /index.php?r=admin%2Fpost%2Findex

echo Url::to(['']);

// a relative route with action ID only: /index.php?r=admin%2Fpost%2Findex

echo Url::to(['index']);

// a relative route: /index.php?r=admin%2Fpost%2Findex

echo Url::to(['post/index']);

// an absolute route: /index.php?r=post%2Findex

echo Url::to(['/post/index']);

// /index.php?r=post%2Findex assume the alias "@posts" is defined as "/post/index"

echo Url::to(['@posts']);

待续。。。。。

**三、请求**

一个应用的请求是用 [yii\web\Request](http://www.yiichina.com/doc/api/2.0/yii-web-request) 对象来表示的，该对象提供了诸如 请求参数（译者注：通常是GET参数或者POST参数）、HTTP头、cookies等信息。 默认情况下，对于一个给定的请求，你可以通过 request [application component](http://www.yiichina.com/doc/guide/2.0/structure-application-components) 应用组件（[yii\web\Request](http://www.yiichina.com/doc/api/2.0/yii-web-request) 类的实例） 获得访问相应的请求对象。

请求参数

要获取请求参数，你可以调用 request 组件的 [get()](http://www.yiichina.com/doc/api/2.0/yii-web-request#get%28%29-detail) 方法和 [post()](http://www.yiichina.com/doc/api/2.0/yii-web-request#post%28%29-detail) 方法。 他们分别返回 $\_GET 和 $\_POST 的值。例如，

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

$get = $request->get();

// 等价于: $get = $\_GET;

$id = $request->get('id');

// 等价于: $id = isset($\_GET['id']) ? $\_GET['id'] : null;

$id = $request->get('id', 1);

// 等价于: $id = isset($\_GET['id']) ? $\_GET['id'] : 1;

$post = $request->post();

// 等价于: $post = $\_POST;

$name = $request->post('name');

// 等价于: $name = isset($\_POST['name']) ? $\_POST['name'] : null;

$name = $request->post('name', '');

// 等价于: $name = isset($\_POST['name']) ? $\_POST['name'] : '';

建议像上面那样通过 request 组件来获取请求参数，而不是 直接访问$\_GET 和 $\_POST。 这使更容易编写测试用例，因为你可以伪造数据来创建一个模拟请求组件。

当实现 [RESTful APIs](http://www.yiichina.com/doc/guide/2.0/rest-quick-start) 接口的时候，你经常需要获取通过PUT， PATCH或者其他的 [request methods](http://www.yiichina.com/doc/guide/2.0/runtime-requests#request-methods) 请求方法提交上来的参数。你可以通过调用 [yii\web\Request::getBodyParam()](http://www.yiichina.com/doc/api/2.0/yii-web-request#getBodyParam%28%29-detail) 方法来获取这些参数。例如，

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

// 返回所有参数

$params = $request->bodyParams;

// 返回参数 "id"

$param = $request->getBodyParam('id');

**info:** 不同于 GET 参数，POST，PUT，PATCH 等等这些提交上来的参数是在请求体中被发送的。 当你通过上面介绍的方法访问这些参数的时候，request 组件会解析这些参数。 你可以通过配置 [yii\web\Request::$parsers](http://www.yiichina.com/doc/api/2.0/yii-web-request#$parsers-detail) 属性来自定义怎样解析这些参数。

请求方法

你可以通过 Yii::$app->request->method 表达式来获取当前请求使用的HTTP方法。 这里还提供了一整套布尔属性用于检测当前请求是某种类型。 例如，

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

if ($request->isAjax) { /\* 该请求是一个 AJAX 请求 \*/ }

if ($request->isGet) { /\* 请求方法是 GET \*/ }

if ($request->isPost) { /\* 请求方法是 POST \*/ }

if ($request->isPut) { /\* 请求方法是 PUT \*/ }

请求URLs

request 组件提供了许多方式来检测当前请求的URL。

假设被请求的URL是 http://example.com/admin/index.php/product?id=100， 你可以像下面描述的那样获取URL的各个部分：

1. yii\web\Request::url：返回 /admin/index.php/product?id=100, 此URL不包括host info部分。
2. yii\web\Request::absoluteUrl：返回 http://example.com/admin/index.php/product?id=100, 包含host infode的整个URL。
3. yii\web\Request::hostInfo：返回 http://example.com, 只有host info部分。
4. yii\web\Request::pathInfo：返回 /product， 这个是入口脚本之后，问号之前（查询字符串）的部分。
5. yii\web\Request::queryString：返回 id=100,问号之后的部分。
6. yii\web\Request::baseUrl：返回 /admin, host info之后， 入口脚本之前的部分。
7. yii\web\Request::scriptUrl：返回 /admin/index.php, 没有path info和查询字符串部分。
8. yii\web\Request::serverName：返回 example.com, URL中的host name。
9. yii\web\Request::serverPort：返回 80, 这是web服务中使用的端口。

HTTP头

你可以通过 yii\web\Request::headers 属性返回的 [header collection](http://www.yiichina.com/doc/api/2.0/yii-web-headercollection) 获取HTTP头信息。 例如，

// $headers 是一个 yii\web\HeaderCollection 对象

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

// 返回 Accept header 值

$accept = $headers->get('Accept');

if ($headers->has('User-Agent')) { /\* 这是一个 User-Agent 头 \*/ }

请求组件也提供了支持快速访问常用头的方法，包括：

1. yii\web\Request::userAgent：返回 User-Agent 头。
2. yii\web\Request::contentType：返回 Content-Type 头的值， Content-Type 是请求体中MIME类型数据。
3. yii\web\Request::acceptableContentTypes：返回用户可接受的内容MIME类型。 返回的类型是按照他们的质量得分来排序的。得分最高的类型将被最先返回。
4. yii\web\Request::acceptableLanguages：返回用户可接受的语言。 返回的语言是按照他们的偏好层次来排序的。第一个参数代表最优先的语言。

客户端信息

你可以通过 yii\web\Request::userHost 和 yii\web\Request::userIP 分别获取host name和客户机的IP地址， 例如，

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

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

**四、响应**

当应用完成处理一个[请求](http://www.yiichina.com/doc/guide/2.0/runtime-requests)后, 会生成一个[response](http://www.yiichina.com/doc/api/2.0/yii-web-response)响应对象并发送给终端用户 响应对象包含的信息有HTTP状态码，HTTP头和主体内容等, 网页应用开发的最终目的本质上就是根据不同的请求构建这些响应对象。

在大多数情况下主要处理继承自 [yii\web\Response](http://www.yiichina.com/doc/api/2.0/yii-web-response) 的 response [应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components)， 尽管如此，Yii也允许你创建你自己的响应对象并发送给终端用户，这方面后续会阐述。

在本节，将会描述如何构建响应和发送给终端用户。

状态码

构建响应时，最先应做的是标识请求是否成功处理的状态，可通过设置 yii\web\Response::statusCode 属性，该属性使用一个有效的 [HTTP 状态码](http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html)。例如，为标识处理已被处理成功， 可设置状态码为200，如下所示：

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

尽管如此，大多数情况下不需要明确设置状态码， 因为 yii\web\Response::statusCode 状态码默认为200， 如果需要指定请求失败，可抛出对应的HTTP异常，如下所示：

throw new \yii\web\NotFoundHttpException;

当[错误处理器](http://www.yiichina.com/doc/guide/2.0/runtime-handling-errors) 捕获到一个异常，会从异常中提取状态码并赋值到响应， 对于上述的 [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception) 对应HTTP 404状态码， 以下为Yii预定义的HTTP异常：

* [yii\web\BadRequestHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-badrequesthttpexception)：状态码 400。
* [yii\web\ConflictHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-conflicthttpexception)：状态码 409。
* [yii\web\ForbiddenHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-forbiddenhttpexception)：状态码 403。
* [yii\web\GoneHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-gonehttpexception)：状态码 410。
* [yii\web\MethodNotAllowedHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-methodnotallowedhttpexception)：状态码 405。
* [yii\web\NotAcceptableHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notacceptablehttpexception)：状态码 406。
* [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception)：状态码 404。
* [yii\web\ServerErrorHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-servererrorhttpexception)：状态码 500。
* [yii\web\TooManyRequestsHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-toomanyrequestshttpexception)：状态码 429。
* [yii\web\UnauthorizedHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-unauthorizedhttpexception)：状态码 401。
* [yii\web\UnsupportedMediaTypeHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-unsupportedmediatypehttpexception)：状态码 415。

如果想抛出的异常不在如上列表中，可创建一个[yii\web\HttpException](http://www.yiichina.com/doc/api/2.0/yii-web-httpexception)异常， 带上状态码抛出，如下：

throw new \yii\web\HttpException(402);

HTTP 头部

可在 response 组件中操控yii\web\Response::headers来发送HTTP头部信息， 例如：

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

// 增加一个 Pragma 头，已存在的Pragma 头不会被覆盖。

$headers->add('Pragma', 'no-cache');

// 设置一个Pragma 头. 任何已存在的Pragma 头都会被丢弃

$headers->set('Pragma', 'no-cache');

// 删除Pragma 头并返回删除的Pragma 头的值到数组

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

**Info:** 头名称是大小写敏感的，在[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 方法调用前新注册的头信息并不会发送给用户。

响应主体

大多是响应应有一个主体存放你想要显示给终端用户的内容。

如果已有格式化好的主体字符串，可赋值到响应的[yii\web\Response::$content](http://www.yiichina.com/doc/api/2.0/yii-web-response#$content-detail)属性， 例如：

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

如果在发送给终端用户之前需要格式化，应设置 [format](http://www.yiichina.com/doc/api/2.0/yii-web-response#$format-detail) 和 [data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail) 属性，[format](http://www.yiichina.com/doc/api/2.0/yii-web-response#$format-detail) 属性指定[data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail)中数据格式化后的样式，例如：

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

$response->format = \yii\web\Response::FORMAT\_JSON;

$response->data = ['message' => 'hello world'];

Yii支持以下可直接使用的格式，每个实现了[formatter](http://www.yiichina.com/doc/api/2.0/yii-web-responseformatterinterface) 类， 可自定义这些格式器或通过配置[yii\web\Response::$formatters](http://www.yiichina.com/doc/api/2.0/yii-web-response#$formatters-detail) 属性来增加格式器。

* [HTML](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_HTML-detail): 通过 [yii\web\HtmlResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-htmlresponseformatter) 来实现.
* [XML](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_XML-detail): 通过 [yii\web\XmlResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-xmlresponseformatter)来实现.
* [JSON](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_JSON-detail): 通过 [yii\web\JsonResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-jsonresponseformatter)来实现.
* [JSONP](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_JSONP-detail): 通过 [yii\web\JsonResponseFormatter](http://www.yiichina.com/doc/api/2.0/yii-web-jsonresponseformatter)来实现.
* [RAW](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_RAW-detail): use this format if you want to send the response directly without applying any formatting.

上述响应主体可明确地被设置，但是在大多数情况下是通过 [操作](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 方法的返回值隐式地设置， 常用场景如下所示：

public function actionIndex()

{

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

}

上述的 index 动作返回 index 视图渲染结果， 返回值会被 response 组件格式化后发送给终端用户。

因为响应格式默认为[HTML](http://www.yiichina.com/doc/api/2.0/yii-web-response#FORMAT_HTML-detail), 只需要在动作方法中返回一个字符串， 如果想使用其他响应格式，应在返回数据前先设置格式，例如：

public function actionInfo()

{

\Yii::$app->response->format = \yii\web\Response::FORMAT\_JSON;

return [

'message' => 'hello world',

'code' => 100,

];

}

如上所述，使用默认的 response 应用组件，也可创建自己的响应对象并发送给终端用户， 可在动作方法中返回该响应对象，如下所示：

public function actionInfo()

{

return \Yii::createObject([

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

'format' => \yii\web\Response::FORMAT\_JSON,

'data' => [

'message' => 'hello world',

'code' => 100,

],

]);

}

**Note:** 如果创建你自己的响应对象，将不能在应用配置中设置 response 组件，尽管如此， 可使用 [依赖注入](http://www.yiichina.com/doc/guide/2.0/concept-di-container) 应用通用配置到你新的响应对象。

浏览器跳转

浏览器跳转依赖于发送一个Location HTTP 头， 因为该功能通常被使用，Yii提供对它提供了特别的支持。

可调用[yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect%28%29-detail) 方法将用户浏览器跳转到一个URL地址，该方法设置合适的 带指定URL的 Location 头并返回它自己为响应对象， 在动作的方法中，可调用缩写版[yii\web\Controller::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-controller#redirect%28%29-detail)，例如：

public function actionOld()

{

return $this->redirect('http://example.com/new', 301);

}

在如上代码中，动作的方法返回redirect() 方法的结果，如前所述， 动作的方法返回的响应对象会被当总响应发送给终端用户。

除了动作方法外，可直接调用[yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect%28%29-detail) 再调用 [yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 方法来确保没有其他内容追加到响应中。

\Yii::$app->response->redirect('http://example.com/new', 301)->send();

**Info:** [yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect%28%29-detail) 方法默认会设置响应状态码为302，该状态码会告诉浏览器请求的资源 临时 放在另一个URI地址上， 可传递一个301状态码告知浏览器请求的资源已经 永久 重定向到新的URId地址。

如果当前请求为AJAX 请求， 发送一个 Location 头不会自动使浏览器跳转，为解决这个问题， [yii\web\Response::redirect()](http://www.yiichina.com/doc/api/2.0/yii-web-response#redirect%28%29-detail) 方法设置一个值为要跳转的URL的X-Redirect 头， 在客户端可编写JavaScript 代码读取该头部值然后让浏览器跳转对应的URL。

**Info:** Yii 配备了一个yii.js JavaScript 文件提供常用JavaScript功能，包括基于X-Redirect头的浏览器跳转， 因此，如果你使用该JavaScript 文件(通过[yii\web\YiiAsset](http://www.yiichina.com/doc/api/2.0/yii-web-yiiasset) 资源包注册)， 就不需要编写AJAX跳转的代码。

发送文件

和浏览器跳转类似，文件发送是另一个依赖指定HTTP头的功能， Yii提供方法集合来支持各种文件发送需求，它们对HTTP头都有内置的支持。

这些方法都将响应对象作为返回值，如果要发送的文件非常大，应考虑使用 [yii\web\Response::sendStreamAsFile()](http://www.yiichina.com/doc/api/2.0/yii-web-response#sendStreamAsFile%28%29-detail) 因为它更节约内存， 以下示例显示在控制器动作中如何发送文件：

public function actionDownload()

{

return \Yii::$app->response->sendFile('path/to/file.txt');

}

如果不是在动作方法中调用文件发送方法， 在后面还应调用 [yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 没有其他内容追加到响应中。

\Yii::$app->response->sendFile('path/to/file.txt')->send();

一些浏览器提供特殊的名为X-Sendfile的文件发送功能， 原理为将请求跳转到服务器上的文件， Web应用可在服务器发送文件前结束，为使用该功能， 可调用[yii\web\Response::xSendFile()](http://www.yiichina.com/doc/api/2.0/yii-web-response#xSendFile%28%29-detail)， 如下简要列出一些常用Web服务器如何启用X-Sendfile 功能：

* Apache: [X-Sendfile](http://tn123.org/mod_xsendfile)
* Lighttpd v1.4: [X-LIGHTTPD-send-file](http://redmine.lighttpd.net/projects/lighttpd/wiki/X-LIGHTTPD-send-file)
* Lighttpd v1.5: [X-Sendfile](http://redmine.lighttpd.net/projects/lighttpd/wiki/X-LIGHTTPD-send-file)
* Nginx: [X-Accel-Redirect](http://wiki.nginx.org/XSendfile)
* Cherokee: [X-Sendfile and X-Accel-Redirect](http://www.cherokee-project.com/doc/other_goodies.html#x-sendfile)

发送响应

在[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 方法调用前响应中的内容不会发送给用户， 该方法默认在[yii\base\Application::run()](http://www.yiichina.com/doc/api/2.0/yii-base-application#run%28%29-detail) 结尾自动调用，尽管如此，可以明确调用该方法强制立即发送响应。

[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 方法使用以下步骤来发送响应：

1. 触发 [yii\web\Response::EVENT\_BEFORE\_SEND](http://www.yiichina.com/doc/api/2.0/yii-web-response#EVENT_BEFORE_SEND-detail) 事件.
2. 调用 [yii\web\Response::prepare()](http://www.yiichina.com/doc/api/2.0/yii-web-response#prepare%28%29-detail) 来格式化 [response data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail) 为 [response content](http://www.yiichina.com/doc/api/2.0/yii-web-response#$content-detail).
3. 触发 [yii\web\Response::EVENT\_AFTER\_PREPARE](http://www.yiichina.com/doc/api/2.0/yii-web-response#EVENT_AFTER_PREPARE-detail) 事件.
4. 调用 [yii\web\Response::sendHeaders()](http://www.yiichina.com/doc/api/2.0/yii-web-response#sendHeaders%28%29-detail) 来发送注册的HTTP头
5. 调用 [yii\web\Response::sendContent()](http://www.yiichina.com/doc/api/2.0/yii-web-response#sendContent%28%29-detail) 来发送响应主体内容
6. 触发 [yii\web\Response::EVENT\_AFTER\_SEND](http://www.yiichina.com/doc/api/2.0/yii-web-response#EVENT_AFTER_SEND-detail) 事件.

一旦[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 方法被执行后，其他地方调用该方法会被忽略， 这意味着一旦响应发出后，就不能再追加其他内容。

如你所见[yii\web\Response::send()](http://www.yiichina.com/doc/api/2.0/yii-web-response#send%28%29-detail) 触发了几个实用的事件， 通过响应这些事件可调整或包装响应。

**五、Sessions 和 Cookies**

Sessions 和 cookies 允许数据在多次请求中保持， 在纯PHP中，可以分别使用全局变量$\_SESSION 和$\_COOKIE 来访问，Yii将session和cookie封装成对象并增加一些功能， 可通过面向对象方式访问它们。

Sessions

和 [请求](http://www.yiichina.com/doc/guide/2.0/runtime-requests) 和 [响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses)类似， 默认可通过为[yii\web\Session](http://www.yiichina.com/doc/api/2.0/yii-web-session) 实例的session [应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components) 来访问sessions。

开启和关闭 Sessions

可使用以下代码来开启和关闭session。

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

// 检查session是否开启

if ($session->isActive) ...

// 开启session

$session->open();

// 关闭session

$session->close();

// 销毁session中所有已注册的数据

$session->destroy();

多次调用[open()](http://www.yiichina.com/doc/api/2.0/yii-web-session#open%28%29-detail) 和[close()](http://www.yiichina.com/doc/api/2.0/yii-web-session#close%28%29-detail) 方法并不会产生错误， 因为方法内部会先检查session是否已经开启。

访问Session数据

可使用如下方式访问session中的数据：

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

// 获取session中的变量值，以下用法是相同的：

$language = $session->get('language');

$language = $session['language'];

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

// 设置一个session变量，以下用法是相同的：

$session->set('language', 'en-US');

$session['language'] = 'en-US';

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

// 删除一个session变量，以下用法是相同的：

$session->remove('language');

unset($session['language']);

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

// 检查session变量是否已存在，以下用法是相同的：

if ($session->has('language')) ...

if (isset($session['language'])) ...

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

// 遍历所有session变量，以下用法是相同的：

foreach ($session as $name => $value) ...

foreach ($\_SESSION as $name => $value) ...

注意: 当使用session组件访问session数据时候， 如果session没有开启会自动开启，这和通过$\_SESSION不同，$\_SESSION要求先执session\_start()。

当session数据为数组时，session组件会限制你直接修改数据中的单元项， 例如：

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

// 如下代码不会生效

$session['captcha']['number'] = 5;

$session['captcha']['lifetime'] = 3600;

// 如下代码会生效：

$session['captcha'] = [

'number' => 5,

'lifetime' => 3600,

];

// 如下代码也会生效：

echo $session['captcha']['lifetime'];

可使用以下任意一个变通方法来解决这个问题：

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

// 直接使用$\_SESSION (确保Yii::$app->session->open() 已经调用)

$\_SESSION['captcha']['number'] = 5;

$\_SESSION['captcha']['lifetime'] = 3600;

// 先获取session数据到一个数组，修改数组的值，然后保存数组到session中

$captcha = $session['captcha'];

$captcha['number'] = 5;

$captcha['lifetime'] = 3600;

$session['captcha'] = $captcha;

// 使用ArrayObject 数组对象代替数组

$session['captcha'] = new \ArrayObject;

...

$session['captcha']['number'] = 5;

$session['captcha']['lifetime'] = 3600;

// 使用带通用前缀的键来存储数组

$session['captcha.number'] = 5;

$session['captcha.lifetime'] = 3600;

为更好的性能和可读性，推荐最后一种方案， 也就是不用存储session变量为数组， 而是将每个数组项变成有相同键前缀的session变量。

自定义Session存储

[yii\web\Session](http://www.yiichina.com/doc/api/2.0/yii-web-session) 类默认存储session数据为文件到服务器上， Yii提供以下session类实现不同的session存储方式：

yii\redis\Session: 存储session数据到以[redis](http://redis.io/) 作为存储媒介中

所有这些session类支持相同的API方法集，因此， 切换到不同的session存储介质不需要修改项目使用session的代码。

注意: 如果通过$\_SESSION访问使用自定义存储介质的session， 需要确保session已经用[yii\web\Session::open()](http://www.yiichina.com/doc/api/2.0/yii-web-session#open%28%29-detail) 开启， 这是因为在该方法中注册自定义session存储处理器。

学习如何配置和使用这些组件类请参考它们的API文档，如下为一个示例 显示如何在应用配置中配置[yii\web\DbSession](http://www.yiichina.com/doc/api/2.0/yii-web-dbsession) 将数据表作为session存储介质。

return [

'components' => [

'session' => [

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

// 'db' => 'mydb', // 数据库连接的应用组件ID，默认为'db'.

// 'sessionTable' => 'my\_session', // session 数据表名，默认为'session'.

],

],

];

也需要创建如下数据库表来存储session数据：

CREATE TABLE session

(

id CHAR(40) NOT NULL PRIMARY KEY,

expire INTEGER,

data BLOB

)

其中'BLOB' 对应你选择的数据库管理系统的BLOB-type类型，以下一些常用数据库管理系统的BLOB类型：

* MySQL: LONGBLOB
* PostgreSQL: BYTEA
* MSSQL: BLOB

注意: 根据php.ini 设置的 session.hash\_function，你需要调整id列的长度， 例如，如果 session.hash\_function=sha256 ， 应使用长度为64而不是40的char类型。

Flash 数据

Flash数据是一种特别的session数据，它一旦在某个请求中设置后， 只会在下次请求中有效，然后该数据就会自动被删除。 常用于实现只需显示给终端用户一次的信息， 如用户提交一个表单后显示确认信息。

可通过session应用组件设置或访问session，例如：

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

// 请求 #1

// 设置一个名为"postDeleted" flash 信息

$session->setFlash('postDeleted', 'You have successfully deleted your post.');

// 请求 #2

// 显示名为"postDeleted" flash 信息

echo $session->getFlash('postDeleted');

// 请求 #3

// $result 为 false，因为flash信息已被自动删除

$result = $session->hasFlash('postDeleted');

和普通session数据类似，可将任意数据存储为flash数据。

当调用[yii\web\Session::setFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#setFlash%28%29-detail)时, 会自动覆盖相同名的已存在的任何数据， 为将数据追加到已存在的相同名flash中，可改为调用[yii\web\Session::addFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#addFlash%28%29-detail)。 例如:

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

// 请求 #1

// 在名称为"alerts"的flash信息增加数据

$session->addFlash('alerts', 'You have successfully deleted your post.');

$session->addFlash('alerts', 'You have successfully added a new friend.');

$session->addFlash('alerts', 'You are promoted.');

// 请求 #2

// $alerts 为名为'alerts'的flash信息，为数组格式

$alerts = $session->getFlash('alerts');

注意: 不要在相同名称的flash数据中使用[yii\web\Session::setFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#setFlash%28%29-detail) 的同时也使用[yii\web\Session::addFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#addFlash%28%29-detail)， 因为后一个防范会自动将flash信息转换为数组以使新的flash数据可追加进来，因此， 当你调用[yii\web\Session::getFlash()](http://www.yiichina.com/doc/api/2.0/yii-web-session#getFlash%28%29-detail)时， 会发现有时获取到一个数组，有时获取到一个字符串， 取决于你调用这两个方法的顺序。

**Tip:** For displaying Flash messages you can use yii\bootstrap\Alert widget in the following way:

echo Alert::widget([

'options' => ['class' => 'alert-info'],

'body' => Yii::$app->session->getFlash('postDeleted'),

]);

Cookies

Yii使用 [yii\web\Cookie](http://www.yiichina.com/doc/api/2.0/yii-web-cookie)对象来代表每个cookie，[yii\web\Request](http://www.yiichina.com/doc/api/2.0/yii-web-request) 和 [yii\web\Response](http://www.yiichina.com/doc/api/2.0/yii-web-response) 通过名为'cookies'的属性维护一个cookie集合， 前者的cookie 集合代表请求提交的cookies， 后者的cookie集合表示发送给用户的cookies。

The part of the application dealing with request and response directly is controller. Therefore, cookies should be read and sent in controller.

读取 Cookies

当前请求的cookie信息可通过如下代码获取：

// 从 "request"组件中获取cookie集合(yii\web\CookieCollection)

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

// 获取名为 "language" cookie 的值，如果不存在，返回默认值"en"

$language = $cookies->getValue('language', 'en');

// 另一种方式获取名为 "language" cookie 的值

if (($cookie = $cookies->get('language')) !== null) {

$language = $cookie->value;

}

// 可将 $cookies当作数组使用

if (isset($cookies['language'])) {

$language = $cookies['language']->value;

}

// 判断是否存在名为"language" 的 cookie

if ($cookies->has('language')) ...

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

发送 Cookies

可使用如下代码发送cookie到终端用户：

// 从"response"组件中获取cookie 集合(yii\web\CookieCollection)

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

// 在要发送的响应中添加一个新的cookie

$cookies->add(new \yii\web\Cookie([

'name' => 'language',

'value' => 'zh-CN',

]));

// 删除一个cookie

$cookies->remove('language');

// 等同于以下删除代码

unset($cookies['language']);

除了上述例子定义的 [name](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$name-detail) 和 [value](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$value-detail) 属性 [yii\web\Cookie](http://www.yiichina.com/doc/api/2.0/yii-web-cookie) 类也定义了其他属性来实现cookie的各种信息，如 [domain](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$domain-detail), [expire](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$expire-detail) 可配置这些属性到cookie中并添加到响应的cookie集合中。

注意: 为安全起见[yii\web\Cookie::$httpOnly](http://www.yiichina.com/doc/api/2.0/yii-web-cookie#$httpOnly-detail) 被设置为true， 这可减少客户端脚本访问受保护cookie（如果浏览器支持）的风险， 更多详情可阅读 [httpOnly wiki article](https://www.owasp.org/index.php/HttpOnly) for more details.

Cookie验证

在上两节中，当通过request 和 response 组件读取和发送cookie时， 你会喜欢扩展的cookie验证的保障安全功能，它能 使cookie不被客户端修改。该功能通过给每个cookie签发一个 哈希字符串来告知服务端cookie是否在客户端被修改， 如果被修改，通过request组件的yii\web\Request::cookiescookie集合访问不到该cookie。

注意: Cookie验证只保护cookie值被修改，如果一个cookie验证失败， 仍然可以通过$\_COOKIE来访问该cookie， 因为这是第三方库对未通过cookie验证自定义的操作方式。

Cookie验证默认启用，可以设置[yii\web\Request::$enableCookieValidation](http://www.yiichina.com/doc/api/2.0/yii-web-request#$enableCookieValidation-detail)属性为false来禁用它， 尽管如此，我们强烈建议启用它。

注意: 直接通过$\_COOKIE 和 setcookie() 读取和发送的Cookie不会被验证。

当使用cookie验证，必须指定[yii\web\Request::$cookieValidationKey](http://www.yiichina.com/doc/api/2.0/yii-web-request#$cookieValidationKey-detail)，它是用来生成s上述的哈希值， 可通过在应用配置中配置request 组件。

return [

'components' => [

'request' => [

'cookieValidationKey' => 'fill in a secret key here',

],

],

];

补充: [cookieValidationKey](http://www.yiichina.com/doc/api/2.0/yii-web-request#$cookieValidationKey-detail) 对你的应用安全很重要， 应只被你信任的人知晓，请不要将它放入版本控制中。

**六、错误处理**

Yii 内置了一个[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler)错误处理器，它使错误处理更方便， Yii错误处理器做以下工作来提升错误处理效果：

* 所有非致命PHP错误（如，警告，提示）会转换成可获取异常；
* 异常和致命的PHP错误会被显示， 在调试模式会显示详细的函数调用栈和源代码行数。
* 支持使用专用的 [控制器动作](http://www.yiichina.com/doc/guide/2.0/structure-controllers#actions) 来显示错误；
* 支持不同的错误响应格式；

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler) 错误处理器默认启用， 可通过在应用的[入口脚本](http://www.yiichina.com/doc/guide/2.0/structure-entry-scripts)中定义常量YII\_ENABLE\_ERROR\_HANDLER来禁用。

使用错误处理器

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler) 注册成一个名称为errorHandler[应用组件](http://www.yiichina.com/doc/guide/2.0/structure-application-components)， 可以在应用配置中配置它类似如下：

return [

'components' => [

'errorHandler' => [

'maxSourceLines' => 20,

],

],

];

使用如上代码，异常页面最多显示20条源代码。

如前所述，错误处理器将所有非致命PHP错误转换成可获取异常， 也就是说可以使用如下代码处理PHP错误：

use Yii;

use yii\base\ErrorException;

try {

10/0;

} catch (ErrorException $e) {

Yii::warning("Division by zero.");

}

// execution continues...

如果你想显示一个错误页面告诉用户请求是无效的或无法处理的， 可简单地抛出一个 [HTTP exception](http://www.yiichina.com/doc/api/2.0/yii-web-httpexception)异常，如 [yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception)。 错误处理器会正确地设置响应的HTTP状态码并使用合适的错误视图页面来显示错误信息。

use yii\web\NotFoundHttpException;

throw new NotFoundHttpException();

自定义错误显示

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler)错误处理器根据常量 YII\_DEBUG的值来调整错误显示，当YII\_DEBUG 为 true (表示在调试模式)， 错误处理器会显示异常以及详细的函数调用栈和源代码行数来帮助调试， 当YII\_DEBUG 为 false，只有错误信息会被显示以防止应用的敏感信息泄漏。

**Info:** 如果异常是继承 [yii\base\UserException](http://www.yiichina.com/doc/api/2.0/yii-base-userexception)， 不管YII\_DEBUG为何值，函数调用栈信息都不会显示， 这是因为这种错误会被认为是用户产生的错误，开发人员不需要去修正。

[error handler](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler) 错误处理器默认使用两个[视图](http://www.yiichina.com/doc/guide/2.0/structure-views)显示错误:

* @yii/views/errorHandler/error.php: 显示不包含函数调用栈信息的错误信息是使用， 当YII\_DEBUG 为 false时，所有错误都使用该视图。
* @yii/views/errorHandler/exception.php: 显示包含函数调用栈信息的错误信息时使用。

可以配置错误处理器的 [errorView](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$errorView-detail) 和 [exceptionView](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$exceptionView-detail) 属性 使用自定义的错误显示视图。

使用错误动作

使用指定的错误[动作](http://www.yiichina.com/doc/guide/2.0/structure-controllers) 来自定义错误显示更方便， 为此，首先配置errorHandler组件的 [errorAction](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$errorAction-detail) 属性， 类似如下：

return [

'components' => [

'errorHandler' => [

'errorAction' => 'site/error',

],

]

];

[errorAction](http://www.yiichina.com/doc/api/2.0/yii-web-errorhandler#$errorAction-detail) 属性使用 [路由](http://www.yiichina.com/doc/guide/2.0/structure-controllers#routes)到一个动作， 上述配置表示不用显示函数调用栈信息的错误会通过执行site/error动作来显示。

可以创建 site/error 动作如下所示：

namespace app\controllers;

use Yii;

use yii\web\Controller;

class SiteController extends Controller

{

public function actions()

{

return [

'error' => [

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

],

];

}

}

上述代码定义 error 动作使用 [yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction) 类， 该类渲染名为error视图来显示错误。

除了使用[yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction), 可定义error 动作使用类似如下的操作方法：

public function actionError()

{

$exception = Yii::$app->errorHandler->exception;

if ($exception !== null) {

return $this->render('error', ['exception' => $exception]);

}

}

现在应创建一个视图文件为views/site/error.php，在该视图文件中，如果错误动作定义为[yii\web\ErrorAction](http://www.yiichina.com/doc/api/2.0/yii-web-erroraction)， 可以访问该动作中定义的如下变量：

* name: 错误名称
* message: 错误信息
* exception: 更多详细信息的异常对象，如HTTP 状态码，错误码， 错误调用栈等。

**Info:** 如果你使用 [基础应用模板](http://www.yiichina.com/doc/guide/2.0/start-installation) 或 [高级应用模板](http://www.yiichina.com/doc/guide/2.0/tutorial-advanced-app), 错误动作和错误视图已经定义好了。

**Note:** If you need to redirect in an error handler, do it the following way:

Yii::$app->getResponse()->redirect($url)->send();

return;

自定义错误格式

错误处理器根据[响应](http://www.yiichina.com/doc/guide/2.0/runtime-responses)设置的格式来显示错误， 如果[response format](http://www.yiichina.com/doc/api/2.0/yii-web-response#$format-detail) 响应格式为html, 会使用错误或异常视图来显示错误信息，如上一小节所述。 对于其他的响应格式，错误处理器会错误信息作为数组赋值给[yii\web\Response::$data](http://www.yiichina.com/doc/api/2.0/yii-web-response#$data-detail)属性， 然后转换到对应的格式， 例如，如果响应格式为json，可以看到如下响应信息：

HTTP/1.1 404 Not Found

Date: Sun, 02 Mar 2014 05:31:43 GMT

Server: Apache/2.2.26 (Unix) DAV/2 PHP/5.4.20 mod\_ssl/2.2.26 OpenSSL/0.9.8y

Transfer-Encoding: chunked

Content-Type: application/json; charset=UTF-8

{

"name": "Not Found Exception",

"message": "The requested resource was not found.",

"code": 0,

"status": 404

}

可在应用配置中响应response组件的beforeSend 事件来自定义错误响应格式。

return [

// ...

'components' => [

'response' => [

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

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

$response = $event->sender;

if ($response->data !== null) {

$response->data = [

'success' => $response->isSuccessful,

'data' => $response->data,

];

$response->statusCode = 200;

}

},

],

],

];

上述代码会重新格式化错误响应，类似如下：

HTTP/1.1 200 OK

Date: Sun, 02 Mar 2014 05:31:43 GMT

Server: Apache/2.2.26 (Unix) DAV/2 PHP/5.4.20 mod\_ssl/2.2.26 OpenSSL/0.9.8y

Transfer-Encoding: chunked

Content-Type: application/json; charset=UTF-8

{

"success": false,

"data": {

"name": "Not Found Exception",

"message": "The requested resource was not found.",

"code": 0,

"status": 404

}

}

**七、日志**

Yii提供了一个强大的日志框架，这个框架具有高度的可定制性和可扩展性。使用这个框架， 你可以轻松地记录各种类型的消息，过滤它们， 并且将它们收集到不同的目标，诸如文件，数据库，邮件。

使用Yii日志框架涉及下面的几个步骤：

* 在你代码里的各个地方记录 [log messages](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-messages)；
* 在应用配置里通过配置 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 来过滤和导出日志消息；
* 检查由不同的目标导出的已过滤的日志消息（例如：[Yii debugger](http://www.yiichina.com/doc/guide/2.0/tool-debugger)）。

在这部分，我们主要描述前两个步骤。

日志消息

记录日志消息就跟调用下面的日志方法一样简单：

* Yii::trace()：记录一条消息去跟踪一段代码是怎样运行的。这主要在开发的时候使用。
* Yii::info()：记录一条消息来传达一些有用的信息。
* Yii::warning()：记录一个警告消息用来指示一些已经发生的意外。
* Yii::error()：记录一个致命的错误，这个错误应该尽快被检查。

这些日志记录方法针对 严重程度 和 类别 来记录日志消息。 它们共享相同的函数签名 function ($message, $category = 'application')，$message代表要被 记录的日志消息，而 $category 是日志消息的类别。在下面的示例代码中，在默认的类别 application 下 记录了一条跟踪消息：

Yii::trace('start calculating average revenue');

**注意:** 日志消息可以是字符串，也可以是复杂的数据，诸如数组或者对象。 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 的义务是正确处理日志消息。默认情况下， 假如一条日志消息不是一个字符串，它将被导出为一个字符串，通过调用 yii\helpers\VarDumper::export()。

为了更好地组织和过滤日志消息，我们建议您为每个日志消息指定一个适当的类别。您可以为类别选择一个分层命名方案， 这将使得 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 在基于它们的分类来过滤消息变得更加容易。 一个简单而高效的命名方案是使用PHP魔术常量 \_\_METHOD\_\_ 作为分类名称。 这种方式也在Yii框架的核心代码中得到应用， 例如，

Yii::trace('start calculating average revenue', \_\_METHOD\_\_);

\_\_METHOD\_\_ 常量计算值作为该常量出现的地方的方法名（完全限定的类名前缀）。 例如，假如上面那行代码在这个方法内被调用，则它将等于字符串 'app\controllers\RevenueController::calculate'。

**注意:** 上面所描述的日志方法实际上是 [logger object](http://www.yiichina.com/doc/api/2.0/yii-log-logger) 对象（一个通过表达式 Yii::getLogger() 可访问的单例） 的方法 [log()](http://www.yiichina.com/doc/api/2.0/yii-log-logger#log%28%29-detail) 的一个快捷方式。当足够的消息被记录或者当应用结束时， 日志对象将会调用一个 [message dispatcher](http://www.yiichina.com/doc/api/2.0/yii-log-dispatcher) 调度对象将已经记录的日志消息发送到已注册的 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 目标中。

日志目标

一个日志目标是一个 [yii\log\Target](http://www.yiichina.com/doc/api/2.0/yii-log-target) 类或者它的子类的实例。 它将通过他们的严重层级和类别来过滤日志消息，然后将它们导出到一些媒介中。 例如，一个 [database target](http://www.yiichina.com/doc/api/2.0/yii-log-dbtarget) 目标导出已经过滤的日志消息到一个数据的表里面， 而一个 [email target](http://www.yiichina.com/doc/api/2.0/yii-log-emailtarget)目标将日志消息导出到指定的邮箱地址里。

在一个应用里，通过配置在应用配置里的 log [application component](http://www.yiichina.com/doc/guide/2.0/structure-application-components) ，你可以注册多个日志目标。 就像下面这样：

return [

// the "log" component must be loaded during bootstrapping time

'bootstrap' => ['log'],

'components' => [

'log' => [

'targets' => [

[

'class' => 'yii\log\DbTarget',

'levels' => ['error', 'warning'],

],

[

'class' => 'yii\log\EmailTarget',

'levels' => ['error'],

'categories' => ['yii\db\\*'],

'message' => [

'from' => ['log@example.com'],

'to' => ['admin@example.com', 'developer@example.com'],

'subject' => 'Database errors at example.com',

],

],

],

],

],

];

**注意:** log 组件必须在 [bootstrapping](http://www.yiichina.com/doc/guide/2.0/runtime-bootstrapping) 期间就被加载，以便于它能够及时调度日志消息到目标里。 这是为什么在上面的代码中，它被列在 bootstrap 数组中的原因。

在上面的代码中，在 [yii\log\Dispatcher::$targets](http://www.yiichina.com/doc/api/2.0/yii-log-dispatcher#$targets-detail) 属性里有两个日志目标被注册：

* 第一个目标选择的是错误和警告层级的消息，并且在数据库表里保存他们；
* 第二个目标选择的是错误层级的消息并且是在以 yii\db\ 开头的分类下，并且在一个邮件里将它们发送到 admin@example.com 和 developer@example.com。

Yii配备了以下的内建日志目标。请参考关于这些类的API文档， 并且学习怎样配置和使用他们。

* [yii\log\DbTarget](http://www.yiichina.com/doc/api/2.0/yii-log-dbtarget)：在数据库表里存储日志消息。
* [yii\log\EmailTarget](http://www.yiichina.com/doc/api/2.0/yii-log-emailtarget)：发送日志消息到预先指定的邮箱地址。
* [yii\log\FileTarget](http://www.yiichina.com/doc/api/2.0/yii-log-filetarget)：保存日志消息到文件中.
* [yii\log\SyslogTarget](http://www.yiichina.com/doc/api/2.0/yii-log-syslogtarget)：通过调用PHP函数 syslog() 将日志消息保存到系统日志里。

下面，我们将描述所有日志目标的公共特性。

消息过滤

对于每一个日志目标，你可以配置它的 yii\log\Target::levels 和 [categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性来指定哪个消息的严重程度和分类目标应该处理。

yii\log\Target::levels 属性是由一个或者若干个以下值组成的数组：

* error：相应的消息通过 Yii::error() 被记录。
* warning：相应的消息通过 Yii::warning() 被记录。
* info：相应的消息通过 Yii::info() 被记录。
* trace：相应的消息通过 Yii::trace() 被记录。
* profile：相应的消息通过 Yii::beginProfile() 和 Yii::endProfile() 被记录。更多细节将在 [Profiling](http://www.yiichina.com/doc/guide/2.0/runtime-logging#performance-profiling) 分段解释。

如果你没有指定 yii\log\Target::levels 的属性， 那就意味着目标将处理 任何 严重程度的消息。

[categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性是一个包含消息分类名称或者模式的数组。 一个目标将只处理那些在这个数组中能够找到对应的分类或者其中一个相匹配的模式的消息。 一个分类模式是一个以星号 \* 结尾的分类名前缀。假如一个分类名与分类模式具有相同的前缀， 那么该分类名将和分类模式相匹配。例如， yii\db\Command::execute 和 yii\db\Command::query 都是作为分类名称运用在 [yii\db\Command](http://www.yiichina.com/doc/api/2.0/yii-db-command) 类来记录日志消息的。 它们都是匹配模式 yii\db\\*。

假如你没有指定 [categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性， 这意味着目标将会处理 任何 分类的消息。

除了通过 [categories](http://www.yiichina.com/doc/api/2.0/yii-log-target#$categories-detail) 属性设置白名单分类，你也可以通过 [except](http://www.yiichina.com/doc/api/2.0/yii-log-target#$except-detail) 属性来设置某些分类作为黑名单。假如一条消息的分类在这个属性中被发现或者是匹配其中一个， 那么它将不会在目标中被处理。

在下面的目标配置中指明了目标应该只处理错误和警告消息，当分类的名称匹配 yii\db\\* 或者是 yii\web\HttpException:\* 的时候， 但是除了 yii\web\HttpException:404。

[

'class' => 'yii\log\FileTarget',

'levels' => ['error', 'warning'],

'categories' => [

'yii\db\\*',

'yii\web\HttpException:\*',

],

'except' => [

'yii\web\HttpException:404',

],

]

**注意:** 当一个HTTP异常通过 [error handler](http://www.yiichina.com/doc/guide/2.0/runtime-handling-errors) 被捕获的时候，一个错误消息将以 yii\web\HttpException:ErrorCode 这样的格式的分类名被记录下来。例如，[yii\web\NotFoundHttpException](http://www.yiichina.com/doc/api/2.0/yii-web-notfoundhttpexception) 将会引发一个分类是 yii\web\HttpException:404 的 错误消息。

消息格式化

日志目标以某种格式导出过滤过的日志消息。例如， 假如你安装一个 [yii\log\FileTarget](http://www.yiichina.com/doc/api/2.0/yii-log-filetarget) 类的日志目标， 你应该能找出一个日志消息类似下面的 runtime/log/app.log 文件：

2014-10-04 18:10:15 [::1][][-][trace][yii\base\Module::getModule] Loading module: debug

默认情况下，日志消息将被格式化，格式化的方式遵循 [yii\log\Target::formatMessage()](http://www.yiichina.com/doc/api/2.0/yii-log-target#formatMessage%28%29-detail)：

Timestamp [IP address][User ID][Session ID][Severity Level][Category] Message Text

你可以通过配置 [yii\log\Target::$prefix](http://www.yiichina.com/doc/api/2.0/yii-log-target#$prefix-detail) 的属性来自定义格式，这个属性是一个PHP可调用体返回的自定义消息前缀。 例如，下面的代码配置了一个日志目标的前缀是每个日志消息中当前用户的ID(IP地址和Session ID被删除是由于隐私的原因)。

[

'class' => 'yii\log\FileTarget',

'prefix' => function ($message) {

$user = Yii::$app->has('user', true) ? Yii::$app->get('user') : null;

$userID = $user ? $user->getId(false) : '-';

return "[$userID]";

}

]

除了消息前缀以外，日志目标也可以追加一些上下文信息到每组日志消息中。 默认情况下，这些全局的PHP变量的值被包含在：$\_GET, $\_POST, $\_FILES, $\_COOKIE,$\_SESSION 和 $\_SERVER 中。 你可以通过配置 [yii\log\Target::$logVars](http://www.yiichina.com/doc/api/2.0/yii-log-target#$logVars-detail) 属性适应这个行为， 这个属性是你想要通过日志目标包含的全局变量名称。 举个例子，下面的日志目标配置指明了只有 $\_SERVER 变量的值将被追加到日志消息中。

[

'class' => 'yii\log\FileTarget',

'logVars' => ['\_SERVER'],

]

你可以将 logVars 配置成一个空数组来完全禁止上下文信息包含。 或者假如你想要实现你自己提供上下文信息的方式， 你可以重写 [yii\log\Target::getContextMessage()](http://www.yiichina.com/doc/api/2.0/yii-log-target#getContextMessage%28%29-detail) 方法。

消息跟踪级别

在开发的时候，通常希望看到每个日志消息来自哪里。这个是能够被实现的，通过配置 log 组件的 yii\log\Dispatcher::traceLevel 属性， 就像下面这样：

return [

'bootstrap' => ['log'],

'components' => [

'log' => [

'traceLevel' => YII\_DEBUG ? 3 : 0,

'targets' => [...],

],

],

];

上面的应用配置设置了 yii\log\Dispatcher::traceLevel 的层级，假如 YII\_DEBUG 开启则是3，否则是0。 这意味着，假如 YII\_DEBUG 开启，每个日志消息在日志消息被记录的时候， 将被追加最多3个调用堆栈层级；假如 YII\_DEBUG 关闭， 那么将没有调用堆栈信息被包含。

**注意:** 获得调用堆栈信息并不是不重要。因此， 你应该只在开发或者调试一个应用的时候使用这个特性。

消息刷新和导出

如上所述，通过 [logger object](http://www.yiichina.com/doc/api/2.0/yii-log-logger) 对象，日志消息被保存在一个数组里。 为了这个数组的内存消耗，当数组积累了一定数量的日志消息， 日志对象每次都将刷新被记录的消息到 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 中。 你可以通过配置 log 组件的 yii\log\Dispatcher::flushInterval 属性来自定义数量：

return [

'bootstrap' => ['log'],

'components' => [

'log' => [

'flushInterval' => 100, // default is 1000

'targets' => [...],

],

],

];

**注意:** 当应用结束的时候，消息刷新也会发生，这样才能确保日志目标能够接收完整的日志消息。

当 [logger object](http://www.yiichina.com/doc/api/2.0/yii-log-logger) 对象刷新日志消息到 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 的时候，它们并 不能立即获取导出的消息。相反，消息导出仅仅在一个日志目标累积了一定数量的过滤消息的时候才会发生。你可以通过配置 个别的 [log targets](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 的 [exportInterval](http://www.yiichina.com/doc/api/2.0/yii-log-target#$exportInterval-detail) 属性来 自定义这个数量，就像下面这样：

[

'class' => 'yii\log\FileTarget',

'exportInterval' => 100, // default is 1000

]

因为刷新和导出层级的设置，默认情况下，当你调用 Yii::trace() 或者任何其他的记录方法，你将不能在日志目标中立即看到日志消息。 这对于一些长期运行的控制台应用来说可能是一个问题。为了让每个日志消息在日志目标中能够立即出现， 你应该设置 yii\log\Dispatcher::flushInterval 和 [exportInterval](http://www.yiichina.com/doc/api/2.0/yii-log-target#$exportInterval-detail) 都为1， 就像下面这样：

return [

'bootstrap' => ['log'],

'components' => [

'log' => [

'flushInterval' => 1,

'targets' => [

[

'class' => 'yii\log\FileTarget',

'exportInterval' => 1,

],

],

],

],

];

**注意:** 频繁的消息刷新和导出将降低你到应用性能。

切换日志目标

你可以通过配置 [enabled](http://www.yiichina.com/doc/api/2.0/yii-log-target#$enabled-detail) 属性来开启或者禁用日志目标。 你可以通过日志目标配置去做，或者是在你的代码中放入下面的PHP申明：

Yii::$app->log->targets['file']->enabled = false;

上面的代码要求您将目标命名为 file，像下面展示的那样， 在 targets 数组中使用使用字符串键：

return [

'bootstrap' => ['log'],

'components' => [

'log' => [

'targets' => [

'file' => [

'class' => 'yii\log\FileTarget',

],

'db' => [

'class' => 'yii\log\DbTarget',

],

],

],

],

];

创建新的目标

创建一个新的日志目标类非常地简单。你主要需要实现 [yii\log\Target::export()](http://www.yiichina.com/doc/api/2.0/yii-log-target#export%28%29-detail) 方法来发送 [yii\log\Target::$messages](http://www.yiichina.com/doc/api/2.0/yii-log-target#$messages-detail) 数组的 内容到一个指定的媒体中。你可以调用 [yii\log\Target::formatMessage()](http://www.yiichina.com/doc/api/2.0/yii-log-target#formatMessage%28%29-detail) 方法去格式化每个消息。 更多细节，你可以参考任何一个包含在Yii发布版中的日志目标类。

性能分析

性能分析是一个特殊的消息记录类型，它通常用在测量某段代码块的时间， 并且找出性能瓶颈是什么。举个例子，[yii\db\Command](http://www.yiichina.com/doc/api/2.0/yii-db-command) 类 使用性能分析找出每个数据库查询的时间。

为了使用性能分析，首先确定需要进行分析的代码块。 然后像下面这样围住每个代码块：

\Yii::beginProfile('myBenchmark');

...code block being profiled...

\Yii::endProfile('myBenchmark');

这里的 myBenchmark 代表一个唯一标记来标识一个代码块。之后当你检查分析结果的时候， 你将使用这个标记来定位对应的代码块所花费的时间。

对于确保 beginProfile 和 endProfile 对能够正确地嵌套，这是很重要的。 例如，

\Yii::beginProfile('block1');

// some code to be profiled

\Yii::beginProfile('block2');

// some other code to be profiled

\Yii::endProfile('block2');

\Yii::endProfile('block1');

假如你漏掉 \Yii::endProfile('block1') 或者切换了 \Yii::endProfile('block1') 和 \Yii::endProfile('block2') 的 顺序，那么性能分析将不会工作。

对于每个被分析的代码块，一个带有严重程度 profile 的日志消息被记录。 你可以配置一个 [log target](http://www.yiichina.com/doc/guide/2.0/runtime-logging#log-targets) 去收集这些 消息，并且导出他们。[Yii debugger](http://www.yiichina.com/doc/guide/2.0/tool-debugger) 有一个内建的性能分析面板能够展示分析结果。