路由：http://hostname/index.php?r=site/say&message=Hello+World

当入口脚本在调用 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处理这个请求。 该过程就被称为引导路由（routing）。

message 被作为一个参数传给 actionSay() 方法，当省略它时，参数将使用默认值代替。

上面 URL 中的参数 r 代表路由，是整个应用级的， 指向特定操作的独立 ID。

路由格式是 控制器 ID/操作 ID。应用接受请求的时候会检查参数， 使用控制器 ID 去确定哪个控制器应该被用来处理请求。 然后相应控制器将使用操作 ID 去确定哪个操作方法将被用来做具体工作。 上述例子中，路由 site/say 将被解析至 SiteController 控制器和其中的 say 操作。 因此 SiteController::actionSay() 方法将被调用处理请求。

注意：与操作一样，一个应用中控制器同样有唯一的 ID。 控制器 ID 和操作 ID 使用同样的命名规则。 控制器的类名源自于控制器 ID，移除了连字符 ，每个单词首字母大写，并加上 Controller 后缀。 例子：控制器 ID post-comment 相当于控制器类名 PostCommentController。

终端用户通过所谓的路由寻找到操作，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;

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 操作。

Yii为开发者提供了路由和URL管理组件：

所谓路由是指URL中用于标识用于处理用户请求的module, controller, action的部分， 一般情况下由 r 查询参数来指定。 如 http://www.digpage.com/index.php?r=post/view&id=100 ， 表示这个请求将由PostController 的 actionView来处理。

同时，Yii也提供了一种美化URL的功能，使得上面的URL可以用一个比较整洁、美观的形式表现出来， 如 http://www.digpage.com/post/view/100 。 这个功能的实现是依赖于一个称为 urlManager 的应用组件。

使用 urlManager 开发者可以解析用户的请求，并指派相应的module, controller和action来进行处理， 还可以根据预义的路由规则，生成需要的URL返回给用户使用。 简而言之，urlManger具有解析请求以便确定指派谁来处理请求和根据路由规则生成URL 2个功能。

===================例子===================================

当前只是理解如何使用路由，具体怎么解析和生成路由，是由路由规则决定的

美化URL

一般情况下，Yii应用生成和接受形如 http://www.digpage.com/index.php?r=post/view&id=100 的URL。这个URL分成几个部分：

表示主机信息的 http://www.digapge.com

表示入口脚本的 index.php

表示路由的 r=post/view

表示普通查询参数的 id=100

其中，主机信息部分从URL来讲，一般是不能少的。当然内部链接可以使用相对路径，这种情况下看似 可以省略，但是User Agent最终发出Request时，也是包含主机信息的。换句话说，Web Server接收并 转交给Yii处理的URL，是完整的、带有主机信息的URL。

而入口脚本 index.php 我们知道，Web Server会将所有的请求都是交由其进行处理。 也就是说，Web Server应当视所有的URL为请求 index.php 脚本。这在 :ref:install 部分我们 已经对Web Server进行过相应配置了。

Yii允许我们不在URL中出现入口脚本 index.php 。

其次，路由信息对于Yii应用而言也必不可少，表明应当使用哪个controller和action来处理请求， 否则Yii只能使用默认的路由来处理请求。这个形式比较固定，采用的是一种类似路径的形式， 一般为 module/controller/action 之类的。

如果将URL省略掉入口脚本，并将路由信息转换成路径，上面的URL就会变成： http://www.digpage.com/post/view?id=100

对于查询参数 id=100 而言，这个URL请求的是编号为100的一个POST， 并执行view操作。那么我们可以再进一步改成 http://www.digpage.com/post/view/100

我们假如所请求的编号100的文章，其标题为 Route ， 那么不妨使用用 http://www.digpage.com/post/view/Route 来访问。

这样的话，干脆再加上 .html 好了。 变成 http://www.digpage.com/post/view/Route.html

Yii有专门的 yii\web\UrlManager 来进行处理，其中：

隐藏入口脚本可以通过 yii\web\UrlManager::showScriptName = false 来实现

路由的路径化可以通过 yii\web\UrlManager::enablePrettyUrl = true 来实现

参数的路径化可以通过路由规则来实现

假后缀(fake suffix) .html 可以通过 yii\web\UrlManager::suffix = '.html' 来实现

路由规则：

路由规则是指 urlManager 用于解析请求或生成URL的规则。 一个路由规则必须实现 yii\web\UrlRuleInterface 接口，这个接口定义了两个方法：

用于解析请求的 yii\web\UrlRuleInterface::parseRequest()

用于生成URL的 yii\web\UrlRuleInterface::createUrl()

Yii中，使用 yii\web\UrlRule 来表示路由规则，一般这个类是足够开发者使用的。 但是，如果开发者想自己实现解析请求或生成URL的逻辑，可以以这个类为基类进行派生， 并重载 parseRuquest() 和 createUrl() 。

以下是配置文件中urlManager组件的路由规则配置部分，以几个相对简单、典型的路由规则的为例， 先有个感性认识:

'rules' => [

// 为路由指定了一个别名，以 post 的复数形式来表示 post/index 路由

'posts' => 'post/index',

// id 是命名参数，post/100 形式的URL，其实是 post/view&id=100

'post/<id:\d+>' => 'post/view',

// controller action 和 id 以命名参数形式出现

'<controller:(post|comment)>/<id:\d+>/<action:(create|update|delete)>'

=> '<controller>/<action>',

// 包含了 HTTP 方法限定，仅限于DELETE方法

'DELETE <controller:\w+>/<id:\d+>' => '<controller>/delete',

// 需要将 Web Server 配置成可以接收 \*.digpage.com 域名的请求

'http://<user:\w+>.digpage.com/<lang:\w+>/profile' => 'user/profile',

]

只需大致了解上面这个数组用于为urlManager声明路由规则。

数组的键相当于请求（需要解析的或将要生成的），而元素的值则对应的路由， 即 controller/action 。

请求部分可称为pattern，路由部分则可称为route。

yii\web\UrlRule 的代码：

class UrlRule extends Object implements UrlRuleInterface

{

// 用于 $mode 表示路由规则的2种工作模式：仅用于解析请求和仅用于生成URL。

// 任意不为1或2的值均表示两种模式同时适用，

// 一般未设定或为0时即表示两种模式均适用。

const PARSING\_ONLY = 1;

const CREATION\_ONLY = 2;

// 路由规则名称

public $name;

// 用于解析请求或生成URL的模式，通常是正则表达式

public $pattern;

// 用于解析或创建URL时，处理主机信息的部分，如 http://www.digpage.com

public $host;

// 指向controller 和 action 的路由

public $route;

// 以一组键值对数组指定若干GET参数，在当前规则用于解析请求时，

// 这些GET参数会被注入到 $\_GET 中去

public $defaults = [];

// 指定URL的后缀，通常是诸如 ".html" 等，

// 使得一个URL看起来好像指向一个静态页面。

// 如果这个值未设定，使用 UrlManager::suffix 的值。

public $suffix;

// 指定当前规则适用的HTTP方法，如 GET, POST, DELETE 等。

// 可以使用数组表示同时适用于多个方法。

// 如果未设定，表明当前规则适用于所有方法。

// 当然，这个属性仅在解析请求时有效，在生成URL时是无效的。

public $verb;

// 表明当前规则的工作模式，取值可以是 0, PARSING\_ONLY, CREATION\_ONLY。

// 未设定时等同于0。

public $mode;

// 表明URL中的参数是否需要进行url编码，默认是进行。

public $encodeParams = true;

// 用于生成新URL的模板

private $\_template;

// 一个用于匹配路由部分的正则表达式，用于生成URL

private $\_routeRule;

// 用于保存一组匹配参数的正则表达式，用于生成URL

private $\_paramRules = [];

// 保存一组路由中使用的参数

private $\_routeParams = [];

// 初始化

public function init() {...}

// 用于解析请求，由UrlRequestInterface接口要求

public function parseRequest($manager, $request) {...}

// 用于生成URL，由UrlRequestInterface接口要求

public function createUrl($manager, $route, $params) {...}

}

从上面代码看， UrlRule 的属性（可配置项）比较多。

要着重分析一下初始化函数 yii\web\UrlRule::init() ，来加深对这些属性的理解:

public function init()

{

// 一个路由规则必定要有 pattern ，否则是没有意义的，

// 一个什么都没规定的规定，要来何用？

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

throw new InvalidConfigException('UrlRule::pattern must be set.');

}

// 不指定规则匹配后所要指派的路由，Yii怎么知道将请求交给谁来处理？

// 不指定路由，Yii怎么知道这个规则可以为谁创建URL？

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

throw new InvalidConfigException('UrlRule::route must be set.');

}

// 如果定义了一个或多个verb，说明规则仅适用于特定的HTTP方法。

// 既然是HTTP方法，那就要全部大写。

// verb的定义可以是字符串（单一的verb）或数组（单一或多个verb）。

if ($this->verb !== null) {

if (is\_array($this->verb)) {

foreach ($this->verb as $i => $verb) {

$this->verb[$i] = strtoupper($verb);

}

} else {

$this->verb = [strtoupper($this->verb)];

}

}

// 若未指定规则的名称，那么使用最能区别于其他规则的 $pattern

// 作为规则的名称

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

$this->name = $this->pattern;

}

// 删除 pattern 两端的 "/"，特别是重复的 "/"，

// 在写 pattern 时，虽然有正则的成分，但不需要在两端加上 "/"，

// 更不能加上 "#" 等其他分隔符

$this->pattern = trim($this->pattern, '/');

// 如果定义了 host ，将 host 部分加在 pattern 前面，作为新的 pattern

if ($this->host !== null) {

// 写入的host末尾如果已经包含有 "/" 则去掉，特别是重复的 "/"

$this->host = rtrim($this->host, '/');

$this->pattern = rtrim($this->host . '/' . $this->pattern, '/');

// 既未定义 host ，pattern 又是空的，那么 pattern 匹配任意字符串。

// 而基于这个pattern的，用于生成的URL的template就是空的，

// 意味着使用该规则生成所有URL都是空的。

// 后续也无需再作其他初始化工作了。

} elseif ($this->pattern === '') {

$this->\_template = '';

$this->pattern = '#^$#u';

return;

// pattern 不是空串，且包含有 '://'，以此认定该pattern包含主机信息

} elseif (($pos = strpos($this->pattern, '://')) !== false) {

// 除 '://' 外，第一个 '/' 之前的内容就是主机信息

if (($pos2 = strpos($this->pattern, '/', $pos + 3)) !== false) {

$this->host = substr($this->pattern, 0, $pos2);

// '://' 后再无其他 '/'，那么整个 pattern 其实就是主机信息

} else {

$this->host = $this->pattern;

}

// pattern 不是空串，且不包含主机信息，两端加上 '/' ，形成一个正则

} else {

$this->pattern = '/' . $this->pattern . '/';

}

// route 也要去掉两头的 '/'

$this->route = trim($this->route, '/');

// 从这里往下，请结合流程图来看

// route 中含有 <参数> ，则将所有参数提取成 [参数 => <参数>]

// 存入 \_routeParams[],

// 如 ['controller' => '<controller>', 'action' => '<action>'],

// 留意这里的短路判断，先使用 strpos()，快速排除无需使用正则的情况

if (strpos($this->route, '<') !== false &&

preg\_match\_all('/<(\w+)>/', $this->route, $matches)) {

foreach ($matches[1] as $name) {

$this->\_routeParams[$name] = "<$name>";

}

}

// 这个 $tr[] 和 $tr2[] 用于字符串的转换

$tr = [

'.' => '\\.',

'\*' => '\\\*',

'$' => '\\$',

'[' => '\\[',

']' => '\\]',

'(' => '\\(',

')' => '\\)',

];

$tr2 = [];

// pattern 中含有 <参数名:参数pattern> ，

// 其中 ':参数pattern' 部分是可选的。

if (preg\_match\_all('/<(\w+):?([^>]+)?>/', $this->pattern, $matches,

PREG\_OFFSET\_CAPTURE | PREG\_SET\_ORDER)) {

foreach ($matches as $match) {

// 获取 “参数名”

$name = $match[1][0];

// 获取 “参数pattern” ，如果未指定，使用 '[^\/]' ，

// 表示匹配除 '/' 外的所有字符

$pattern = isset($match[2][0]) ? $match[2][0] : '[^\/]+';

// 如果 defaults[] 中有同名参数，

if (array\_key\_exists($name, $this->defaults)) {

// $match[0][0] 是整个 <参数名:参数pattern> 串

$length = strlen($match[0][0]);

$offset = $match[0][1];

// pattern 中 <参数名:参数pattern> 两头都有 '/'

if ($offset > 1 && $this->pattern[$offset - 1] === '/'

&& $this->pattern[$offset + $length] === '/') {

// 留意这个 (?P<name>pattern) 正则，这是一个命名分组。

// 仅冠以一个命名供后续引用，使用上与直接的 (pattern) 没有区别

// 见：http://php.net/manual/en/regexp.reference.subpatterns.php

$tr["/<$name>"] = "(/(?P<$name>$pattern))?";

} else {

$tr["<$name>"] = "(?P<$name>$pattern)?";

}

// defaults[]中没有同名参数

} else {

$tr["<$name>"] = "(?P<$name>$pattern)";

}

// routeParams[]中有同名参数

if (isset($this->\_routeParams[$name])) {

$tr2["<$name>"] = "(?P<$name>$pattern)";

// routeParams[]中没有同名参数，则将 参数pattern 存入 \_paramRules[] 中。

// 留意这里是怎么对 参数pattern 进行处理后再保存的。

} else {

$this->\_paramRules[$name] = $pattern === '[^\/]+' ? '' :

"#^$pattern$#u";

}

}

}

// 将 pattern 中所有的 <参数名:参数pattern> 替换成 <参数名> 后作为 \_template

$this->\_template = preg\_replace('/<(\w+):?([^>]+)?>/', '<$1>', $this->pattern);

// 将 \_template 中的特殊字符及字符串使用 tr[] 进行转换，并作为最终的pattern

$this->pattern = '#^' . trim(strtr($this->\_template, $tr), '/') . '$#u';

// 如果指定了 routePrams 还要使用 tr2[] 对 route 进行转换，

// 并作为最终的 \_routeRule

if (!empty($this->\_routeParams)) {

$this->\_routeRule = '#^' . strtr($this->route, $tr2) . '$#u';

}

}

先看init() 的前半部分，这些代码提醒我们：

规则的 $pattern 和 $route 是必须配置的。

规则的名称 $name 和主机信息 $host 在未配置的情况下，可以从 $pattern 来获取。

$pattern 虽然含有正则的成分，但不需要在两端加入 / ，更不能使用 # 等其他分隔符。 Yii会自动为我们加上。

指定 $pattern 为空串，可以使该规则匹配任意的URL。此时基于该规则所生成的所有URL也都是空串。

$pattern 中含有 :\\ 时，Yii会认为其中包含了主机信息。此时就不应当再指定 host 。 否则，Yii会将 host 接在这个 pattern 前，作为新的pattern。这会造成该pattern 两段 :\\ ， 而这显然不是我们要的。

>>>>待续部分看截图 路由.png<<<<<