

# PHP RESTful

REST ( 英文 : Representational State Transfer , 简称REST) , 指的是一组架构约束条件和原则。

符合REST设计风格的Web API称为RESTful API。它从以下三个方面资源进行定义 :

- 直观简短的资源地址 : URI , 比如 : `http://example.com/resources/`。
- 传输的资源 : Web服务接受与返回的互联网媒体类型 , 比如 : JSON , XML , YAM等。
- 对资源的操作 : Web服务在该资源上所支持的一系列请求方法 ( 比如 : POST , GET , PUT或DELETE ) 。

本教程我们将使用 PHP(不用框架) 来创建一个 RESTful web service , 在文章末尾你可以下载本章节使用到的代码。

通过本教程你将学习到以下内容 :

- 创建一个 RESTful Webservice。
- 使用原生 PHP, 不依赖任何框架。
- URI 模式需要遵循 REST 规则。
- RESTful service 接受与返回的格式可以是 JSON, XML等。
- 根据不同情况响应对应的 HTTP 状态码。
- 演示请求头的使用。
- 使用 REST 客户端来测试 RESTful web service。

## RESTful Webservice 实例

以下代码是 RESTful 服务类 **Site.php** :

### 实例

```
<?php
/*
 * 菜鸟教程 RESTful 演示实例
 * RESTful 服务类
 */
Class Site {
private $sites = array(
1 => 'TaoBao',
2 => 'Google',
3 => 'Runoob',
4 => 'Baidu',
5 => 'Weibo',
6 => 'Sina'
);
public function getAllSite(){
return $this->sites;
}
public function getSite($id){
```

```
$site = array($id => ($this->sites[$id]) ? $this->sites[$id] : $this->sites[1]);
return $site;
}
}
?>
```

## RESTful Services URI 映射

RESTful Services URI 应该设置为一个直观简短的资源地址。Apache 服务器的 .htaccess 应设置好对应的 Rewrite 规则。

本实例我们将使用两个 URI 规则：

1、获取所有站点列表：

```
http://localhost/restexample/site/list/
```

2、使用 id 获取指定的站点，以下 URI 为获取 id 为 3 的站点：

```
http://localhost/restexample/site/list/3/
```

项目的 .htaccess 文件配置规则如下所示：

```
# 开启 rewrite 功能
Options +FollowSymlinks
RewriteEngine on

# 重写规则
RewriteRule ^site/list/$ RestController.php?view=all [nc,rsa]
RewriteRule ^site/list/([0-9]+)/$ RestController.php?view=single&id=$1 [nc,rsa]
```

## RESTful Web Service 控制器

在 .htaccess 文件中，我们通过设置参数 'view' 来获取 RestController.php 文件中对应的请求，通过获取 'view' 不同的参数来分发到不同的方法上。RestController.php 文件代码如下：

### 实例

```
<?php
require_once("SiteRestHandler.php");
$view = "";
if(isset($_GET["view"]))
$view = $_GET["view"];
/*
 * RESTful service 控制器
 * URL 映射
 */
switch($view){
case "all":
// 处理 REST Url /site/list/
```

```
$siteRestHandler = new SiteRestHandler();
$siteRestHandler->getAllSites();
break;
case "single":
// 处理 REST Url /site/show/<id>/
$siteRestHandler = new SiteRestHandler();
$siteRestHandler->getSite($_GET["id"]);
break;
case "" :
//404 - not found;
break;
}
?>
```

## 简单的 RESTful 基础类

以下提供了 RESTful 的一个基类，用于处理响应请求的 HTTP 状态码，**SimpleRest.php** 文件代码如下：

### 实例

```
<?php
/*
 * 一个简单的 RESTful web services 基类
 * 我们可以基于这个类来扩展需求
 */
class SimpleRest {
private $httpVersion = "HTTP/1.1";
public function setHttpHeaders($contentType, $statusCode){
$statusMessage = $this -> getHttpStatusMessage($statusCode);
header($this->httpVersion. " ". $statusCode . " ". $statusMessage);
header("Content-Type:". $contentType);
}
public function getHttpStatusMessage($statusCode){
$httpStatus = array(
100 => 'Continue',
101 => 'Switching Protocols',
200 => 'OK',
201 => 'Created',
202 => 'Accepted',
203 => 'Non-Authoritative Information',
204 => 'No Content',
205 => 'Reset Content',
206 => 'Partial Content',
300 => 'Multiple Choices',
301 => 'Moved Permanently',
302 => 'Found',
303 => 'See Other',
304 => 'Not Modified',
305 => 'Use Proxy',
306 => '(Unused)',
307 => 'Temporary Redirect',
400 => 'Bad Request',
401 => 'Unauthorized',
402 => 'Payment Required',
403 => 'Forbidden',
```

```

404 => 'Not Found',
405 => 'Method Not Allowed',
406 => 'Not Acceptable',
407 => 'Proxy Authentication Required',
408 => 'Request Timeout',
409 => 'Conflict',
410 => 'Gone',
411 => 'Length Required',
412 => 'Precondition Failed',
413 => 'Request Entity Too Large',
414 => 'Request-URI Too Long',
415 => 'Unsupported Media Type',
416 => 'Requested Range Not Satisfiable',
417 => 'Expectation Failed',
500 => 'Internal Server Error',
501 => 'Not Implemented',
502 => 'Bad Gateway',
503 => 'Service Unavailable',
504 => 'Gateway Timeout',
505 => 'HTTP Version Not Supported');
return ($httpStatus[$statusCode]) ? $httpStatus[$statusCode] : $status[500];
}
}
?>

```

## RESTful Web Service 处理类

以下是一个 RESTful Web Service 处理类 SiteRestHandler.php，继承了上面我们提供的 RESTful 基类，类中通过判断请求的参数来决定返回的 HTTP 状态码及数据格式，实例中我们提供了三种数据格式："application/json"、"application/xml" 或 "text/html"：

SiteRestHandler.php 文件代码如下：

### 实例

```

<?php
require_once("SimpleRest.php");
require_once("Site.php");
class SiteRestHandler extends SimpleRest {
function getAllSites() {
    $site = new Site();
    $rawData = $site->getAllSite();
    if(empty($rawData)) {
        $statusCode = 404;
        $rawData = array('error' => 'No sites found!');
    } else {
        $statusCode = 200;
    }
    $requestContentType = $_SERVER['HTTP_ACCEPT'];
    $this->setHttpHeaders($requestContentType, $statusCode);
    if(strpos($requestContentType, 'application/json') !== false){
        $response = $this->encodeJson($rawData);
        echo $response;
    } else if(strpos($requestContentType, 'text/html') !== false){

```

```

$response = $this->encodeHtml($rawData);
echo $response;
} else if(strpos($requestContentType, 'application/xml') !== false){
$response = $this->encodeXml($rawData);
echo $response;
}
}

public function encodeHtml($responseData) {
$htmlResponse = "<table border='1'>";
foreach($responseData as $key=>$value) {
$htmlResponse .= "<tr><td>". $key. "</td><td>". $value. "</td></tr>";
}
$htmlResponse .= "</table>";
return $htmlResponse;
}

public function encodeJson($responseData) {
$jsonResponse = json_encode($responseData);
return $jsonResponse;
}

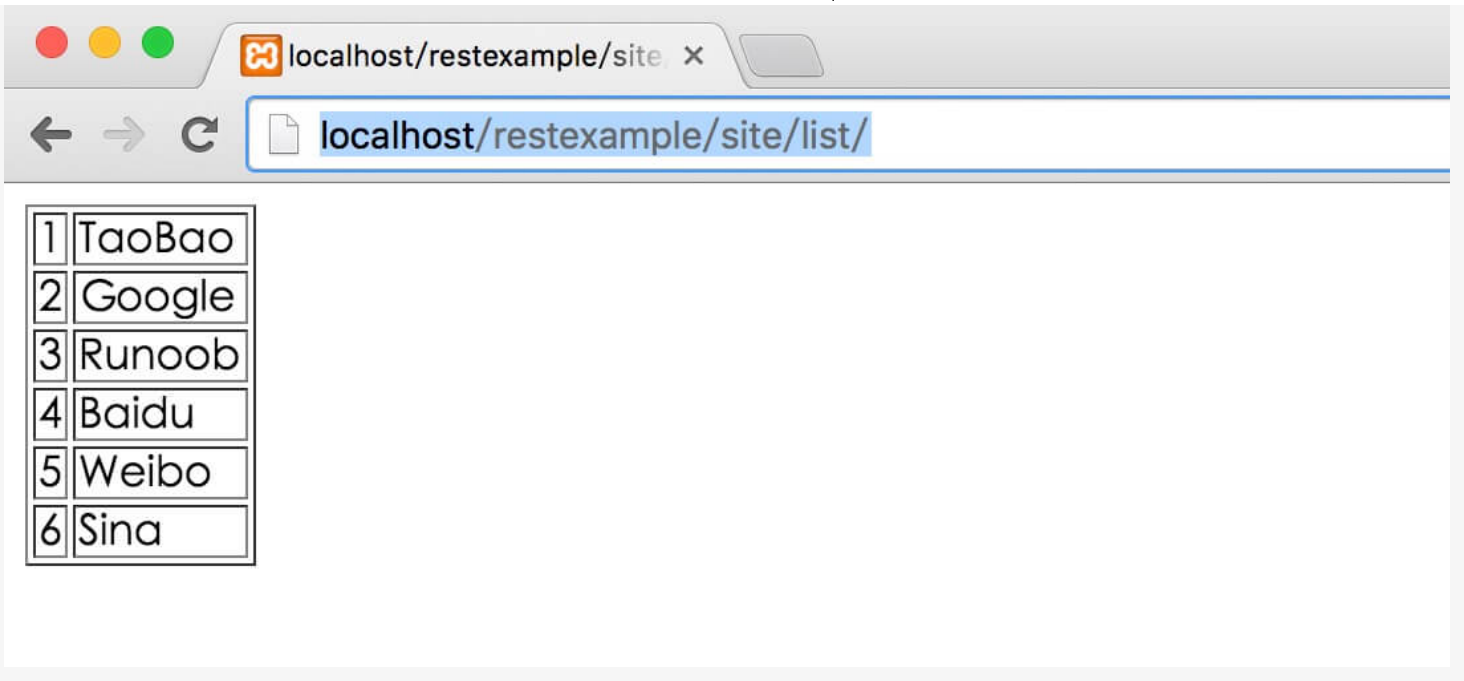
public function encodeXml($responseData) {
// 创建 SimpleXMLElement 对象
$xml = new SimpleXMLElement('<?xml version="1.0"?><site></site>');
foreach($responseData as $key=>$value) {
$xml->addChild($key, $value);
}
return $xml->asXML();
}

public function getSite($id) {
$site = new Site();
$rawData = $site->getSite($id);
if(empty($rawData)) {
$statuscode = 404;
$rawData = array('error' => 'No sites found!');
} else {
$statuscode = 200;
}

$requestContentType = $_SERVER['HTTP_ACCEPT'];
$this->setHttpHeaders($requestContentType, $statusCode);
if(strpos($requestContentType, 'application/json') !== false){
$response = $this->encodeJson($rawData);
echo $response;
} else if(strpos($requestContentType, 'text/html') !== false){
$response = $this->encodeHtml($rawData);
echo $response;
} else if(strpos($requestContentType, 'application/xml') !== false){
$response = $this->encodeXml($rawData);
echo $response;
}
}
}
?>

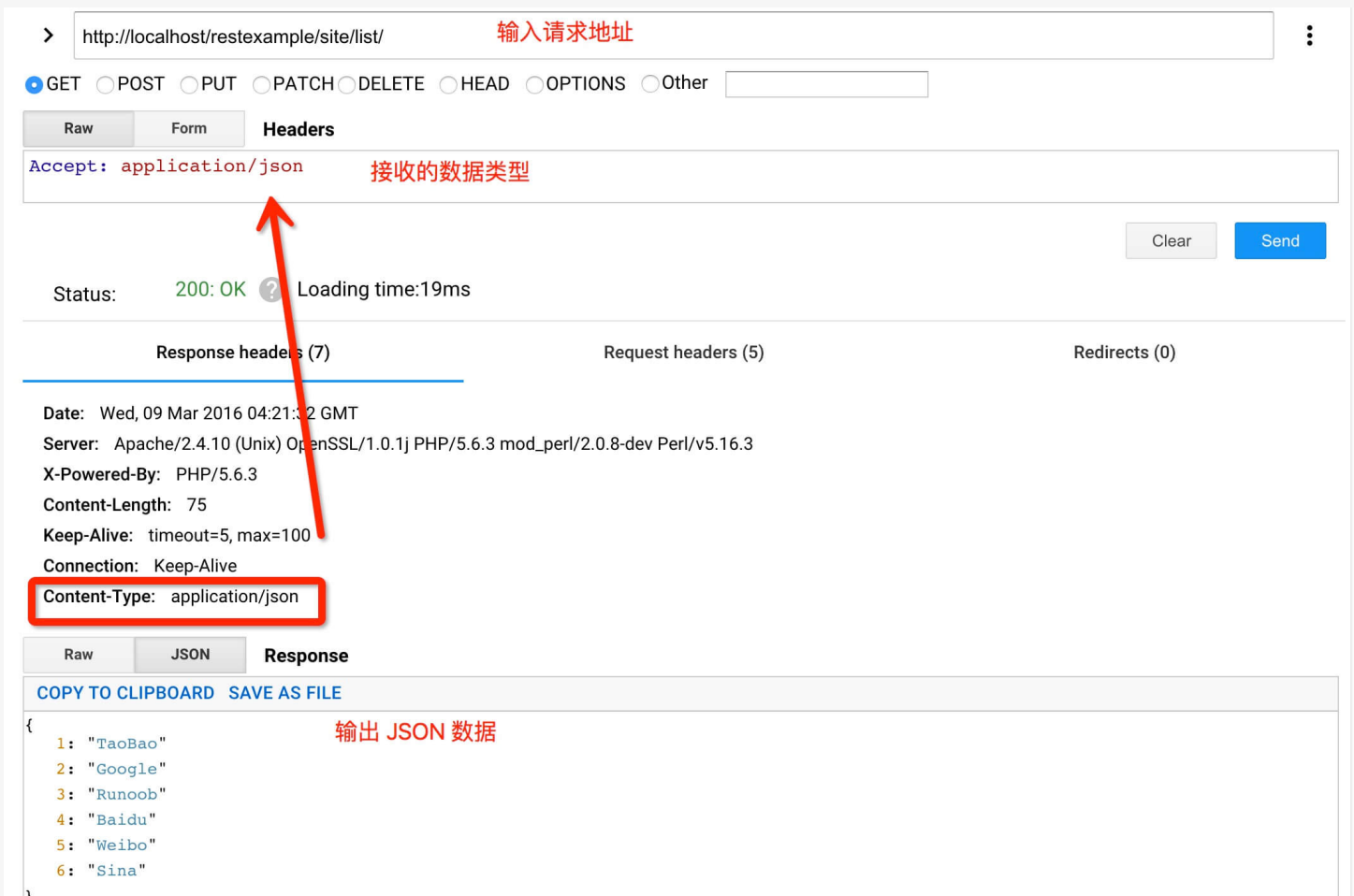
```

接下来我们通过 <http://localhost/restexample/site/list/> 访问，输出结果如下：



## RESTful Web Service 客户端

接下来我们可以使用 Google Chrome 浏览器的 "Advance Rest Client" 作为 RESTful Web Service 客户端来请求我们的服务。实例中请求 `http://localhost/restexample/site/list/` 地址，接收数据类型为 **Accept: application/json**



请求 id 为 3 的站点 Runoob(菜鸟教程)，访问地址为 `http://localhost/restexample/site/list/3/`，

>

http://localhost/restexample/site/list/3/

⋮

☒ GET ☐ POST ☐ PUT ☐ PATCH ☐ DELETE ☐ HEAD ☐ OPTIONS ☐ Other

RawFormHeaders

Accept: application/xml

返回 xml 格式数据

Status: 200: OK

Loading time:23ms

ClearSend

Response headers (7)

Request headers (5)

Redirects (0)

Date: Wed, 09 Mar 2016 04:31:07 GMT

Server: Apache/2.4.10 (Unix) OpenSSL/1.0.1j PHP/5.6.3 mod\_perl/2.0.8-dev Perl/v5.16.3

X-Powered-By: PHP/5.6.3

Content-Length: 49

Keep-Alive: timeout=5, max=100

Connection: Keep-Alive

Content-Type: application/xml

RawParsedResponse

OPEN OUTPUT IN NEW WINDOW COPY TO CLIPBOARD SAVE AS FILE OPEN IN JSON TAB

<?xml version="1.0"?>  
<site><3>Runoob</3></site>

以 xml 格式输出数据

Code highlighting thanks to CODE MIRROR

源码下载

实例中使用到的代码可点击以下按钮下载：

↓

源码下载

点我分享笔记