身份验证 #

目录
REST API 密钥
在 WordPress 管理界面生成 API 密钥
使用应用认证端点自动生成 API 密钥
URL parameters
创建认证端点 URL
注意事项
Authentication over HTTPS
通过 HTTP 进行身份验证
创建签名
收集请求方法和 URL
收集参数
创建签名基础字符串
生成签名
OAuth 使用技巧

WooCommerce 提供了两种与 WP REST API 进行身份验证的方式。您也可以使用任何 WP REST API 身份验证 插件或方法进行身份验证。

REST API 密钥
预生成的密钥可用于验证 REST API 端点的使用。新密钥可以通过 WordPress
管理界面生成，也可以通过端点自动生成。

在 WordPress 管理界面生成 API 密钥
要为特定 WordPress 用户创建或管理密钥，请前往 WooCommerce > 设置 > API > 密钥 /应用。

WooCommerce REST API 密钥设置

点击 “添加密钥 ”按钮。在下一个屏幕中，添加描述并选择要为其生成密钥的 WordPress 用户。使用生成的密钥进行 REST API 调用将遵循该用户的 WordPress 角色和权限。

为此 REST API 密钥选择访问级别，可以是 读取 访问、 写入 访问或 读取 /写入 访问。然后点击 “生成 API 密钥 ”按钮， WooCommerce 将为所选用户生成 REST API 密钥。

创建新的 REST API 密钥

密钥生成后，您应该会看到两个新密钥、一个二维码和一个 “撤销 API 密钥 ” 按钮。这两个密钥分别是您的消费者密钥和消费者密钥。

已生成的 REST API 密钥

如果与 API 密钥关联的 WordPress 用户被删除，该 API 密钥将停止工作。 API 密钥不会转移给其他用户。

使用应用认证端点自动生成 API 密钥

任何应用均可使用此端点，以 允许用户为您的应用生成 *API* 密钥。这使得与 WooCommerce API 的集成更加简便，因为用户只需通过一个 URL 授予您的应用访问权限。在重定向回您的应用后， API 密钥将通过一个单独的 POST 请求发送回来。

下图展示了其工作原理：

认证端点流程

此端点专用于为用户生成 API 密钥，并促进 WooCommerce REST API 与应用之间的集成。此端点绝不可用作客户的登录方式。

URL parameters

Parameter	Type	Description
`app_name`	string	Your APP name *mandatory*
`scope`	string	Level of access. Available: `read`, `write` and `read_write` *mandatory*
`user_id`	string	User ID in your APP. For your internal reference, used when the user is redirected back to your APP. NOT THE USER ID IN WOOCOMMERCE *mandatory*
`return_url`	string	URL the user will be redirected to after authentication *mandatory*
`callback_url`	string	URL that will receive the generated API key. Note: this URL should be over **HTTPS** *mandatory*

□□□□□□ URL

□□□□□ `/wc-auth/v1/authorize` □□□□□□□□□□□□□□□□□□□□

> “□□□□□□ URL □□□□

```
# Bash □□
STORE_URL='http://example.com'
ENDPOINT='/wc-auth/v1/authorize'
PARAMS="app_name=My App Name&scope=read_write&user_id=123&return_url=http://app.com/return-page&callback_url=https://app.com/callback-endpoint"
QUERY_STRING="$(perl -MURI::Escape -e 'print uri_escape($ARGV[0]);' "$PARAMS")"
QUERY_STRING=$(echo $QUERY_STRING | sed -e "s/%20/\+/g" -e "s/%3D/\=/g" -e "s/%26/\&/g")

echo "$STORE_URL$ENDPOINT?$QUERY_STRING"
```

```
const querystring = require('querystring');

const store_url = 'http://example.com';
const endpoint = '/wc-auth/v1/authorize';
const params = {
  app_name: 'My App Name',
  scope: 'read_write',
  user_id: 123,
  return_url: 'http://app.com/return-page',
  callback_url: 'https://app.com/callback-endpoint'
};
const query_string = querystring.stringify(params).replace(/%20/g, '+');

console.log(store_url + endpoint + '?' + query_string);
```

```
<?php
$store_url = 'http://example.com';
$endpoint = '/wc-auth/v1/authorize';
$params = [
    'app_name' => 'My App Name',
    'scope' => 'write',
    'user_id' => 123,
```

```
    'return_url' => 'http://app.com',
    'callback_url' => 'https://app.com'
];
$query_string = http_build_query( $params );

echo $store_url . $endpoint . '?' . $query_string;
?>
```

```
from urllib.parse import urlencode

store_url = 'http://example.com'
endpoint = '/wc-auth/v1/authorize'
params = {
    "app_name": "My App Name",
    "scope": "read_write",
    "user_id": 123,
    "return_url": "http://app.com/return-page",
    "callback_url": "https://app.com/callback-endpoint"
}
query_string = urlencode(params)

print("%s%s?%s" % (store_url, endpoint, query_string))
```

```
require "uri"

store_url = 'http://example.com'
endpoint = '/wc-auth/v1/authorize'
params = {
  app_name: "My App Name",
  scope: "read_write",
  user_id: 123,
  return_url: "http://app.com/return-page",
  callback_url: "https://app.com/callback-endpoint"
}
query_string = URI.encode_www_form(params)

puts "#{store_url}#{endpoint}?#{query_string}"
```

> 随 API 密钥一起发布的 JSON 示例

```
{
    "key_id": 1,
    "user_id": 123,
    "consumer_key": "ck_xxxxxxxxxxxxxxxx",
    "consumer_secret": "cs_xxxxxxxxxxxxxxxx",
    "key_permissions": "read_write"
}
```

用户将看到的界面示例：

认证端点示例

注意事项

- 使用 `return_url` 重定向用户时，还会以查询字符串形式发送 `success` 和 `user_id` 参数。
- `success` 参数在用户拒绝时发送 `0`，认证成功时发送 `1`。
- 当用户被重定向回 (`return_url`) 时，使用 `user_id` 来识别用户，并记得在认证后向您的 `callback_url` 发送 POST 请求时保存 API 密钥。
- 认证端点将以 JSON 格式向 `callback_url` 发送 API 密钥，请注意某些语言（如 PHP）不会在 `$_POST` 全局变量中显示它。在 PHP 中，您可以使用 `$HTTP_RAW_POST_DATA`（适用于旧版 PHP）或 `file_get_contents('php://input');` 来访问。
- 生成的 URL 必须对所有查询字符串值进行编码。

Authentication over HTTPS

You may use HTTP Basic Auth by providing the REST API Consumer Key as the username and the REST API Consumer Secret as the password.

> HTTP Basic Auth example

```
curl https://www.example.com/wp-json/wc/v2/orders \
    -u consumer_key:consumer_secret
```

```
const WooCommerceRestApi = require("@woocommerce/woocommerce-rest-api").default;
// import WooCommerceRestApi from "@woocommerce/woocommerce-rest-api"; // Supports ESM
```

```
const WooCommerce = new WooCommerceRestApi({
  url: 'https://example.com',
  consumerKey: 'consumer_key',
  consumerSecret: 'consumer_secret',
  version: 'wc/v2'
});
```

```
<?php
require __DIR__ . '/vendor/autoload.php';

use Automattic\WooCommerce\Client;

$woocommerce = new Client(
    'https://example.com',
    'consumer_key',
    'consumer_secret',
    [
        'wp_api' => true,
        'version' => 'wc/v2'
    ]
);
?>
```

```
from woocommerce import API

wcapi = API(
    url="https://example.com",
    consumer_key="consumer_key",
    consumer_secret="consumer_secret",
    wp_api=True,
    version="wc/v2"
)
```

```
require "woocommerce_api"

woocommerce = WooCommerce::API.new(
  "https://example.com",
  "consumer_key",
  "consumer_secret",
  {
```

```
    wp_json: true,
    version: "wc/v2"
  }
)
```

Occasionally some servers may not parse the Authorization header correctly (if you see a "Consumer key is missing" error when authenticating over SSL, you have a server issue). In this case, you may provide the consumer key/secret as query string parameters instead.

> Example for servers that not properly parse the Authorization header:

```
curl https://www.example.com/wp-json/wc/v2/orders?consumer_key=123&consumer_secret=abc
```

```
const WooCommerceRestApi = require("@woocommerce/woocommerce-rest-api").default;
// import WooCommerceRestApi from "@woocommerce/woocommerce-rest-api"; // Supports ESM

const WooCommerce = new WooCommerceRestApi({
  url: 'https://example.com',
  consumerKey: 'consumer_key',
  consumerSecret: 'consumer_secret',
  version: 'wc/v2',
  queryStringAuth: true // Force Basic Authentication as query string true and using under
HTTPS
});
```

```
<?php
require __DIR__ . '/vendor/autoload.php';

use Automattic\WooCommerce\Client;

$woocommerce = new Client(
    'https://example.com',
    'consumer_key',
    'consumer_secret',
    [
        'wp_api' => true,
        'version' => 'wc/v2',
        'query_string_auth' => true // Force Basic Authentication as query string true and
using under HTTPS
```

```
    ]
);
?>
```

```
from woocommerce import API

wcapi = API(
    url="https://example.com",
    consumer_key="consumer_key",
    consumer_secret="consumer_secret",
    wp_api=True,
    version="wc/v2",
    query_string_auth=True // Force Basic Authentication as query string true and using under
HTTPS
)
```

```
require "woocommerce_api"

woocommerce = WooCommerce::API.new(
  "https://example.com",
  "consumer_key",
  "consumer_secret",
  {
    wp_json: true,
    version: "wc/v2",
    query_string_auth: true // Force Basic Authentication as query string true and using under
HTTPS
  }
)
```

□□ HTTP □□□□□

□□□□□ OAuth 1.0a "□□□ □□□" □□□□ □□□□ REST API □□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□ OAuth 1.0a □□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□

收集请求方法和 URL

首先需要确定请求使用的 HTTP 方法以及请求的 URL。

HTTP 方法 在本例中为 `GET`。

请求 **URL** 是您要发送请求的端点，例如 `http://www.example.com/wp-json/wc/v2/orders`。

收集参数

收集并规范化您的参数。这包括 `oauth_signature` 本身外的所有 `oauth_*` 参数。

这些值需要编码成一个字符串，后续将使用该字符串。构建此字符串的过程非常具体：

1. 对每个待签名的键和值进行 百分比编码。
2. 按编码后的键对参数列表进行字母顺序排序。
3. 对于每个键 /值对：
   - 将编码后的键追加到输出字符串。
   - 将 `=` 字符追加到输出字符串。
   - 将编码后的值追加到输出字符串。
   - 如果还有更多键 /值对剩余，则将 `&` 字符追加到输出字符串。

例如，在 PHP 中进行百分比编码时，您应使用 `rawurlencode()`。

例如，在 PHP 中对参数排序时，您应使用 `uksort( $params, 'strcmp' )`。

> 参数示例：

```
oauth_consumer_key=abc123&oauth_signature_method=HMAC-SHA1
```

创建签名基础字符串

目前收集到的上述值必须组合成一个字符串，签名将基于此字符串生成。这在 OAuth 规范中称为签名基础字符串。

要将 HTTP 方法、请求 URL 和参数字符串编码为单个字符串：

1. 将输出字符串设置为大写的 **HTTP** 方法。
2. 在输出字符串后追加 `&` 字符。
3. 对 URL 进行 百分比编码并追加到输出字符串。
4. 在输出字符串后追加 `&` 字符。
5. 对参数字符串进行 百分比编码并追加到输出字符串。

> 签名基础字符串示例：

```
GET&http%3A%2F%2Fwww.example.com%2Fwp-
json%2Fwc%2Fv2%2Forders&oauth_consumer_key%3Dabc123%26oauth_signature_method%3DHMAC-SHA1
```

生成签名

使用 签名基础字符串 和您的消费者密钥，通过 HMAC-SHA1 哈希算法生成签名，消费者密钥前需添加一个 `&` 字符。

在 PHP 中，您可以使用 hash_hmac 函数。

HMAC-SHA1 或 HMAC-SHA256 是唯一被接受的哈希算法。

如果您在生成正确签名时遇到问题，请检查您要签名的字符串是否存在编码错误。 身份验证源代码_____
也有助于理解如何正确生成签名。

OAuth 使用技巧

- OAuth 参数可作为查询字符串参数添加，或包含在 Authorization 头信息中。
- 请注意，在 WordPress 中没有可靠的跨平台方法来获取原始请求头信息，因此在某些情况下，查询字符串可能更可靠。
- 必需的参数包括：`oauth_consumer_key`、`oauth_timestamp`、`oauth_nonce`、`oauth_signature` 和 `oauth_signature_method`。`oauth_version` 不是必需的，应省略。
- OAuth nonce 可以是任何随机生成的 32 字符（推荐）字符串，且对消费者密钥唯一。
- OAuth 时间戳应为请求时的 Unix 时间戳。 REST API 将拒绝包含时间戳超出 15 分钟窗口的请求，以防止重放攻击。
- 在构建用于签名的 base string 时，必须使用索引提供的商店 URL，因为服务器将使用此 URL 。（例如，如果商店 URL 包含 `www` 子域名，则应在请求中使用它）
- 请注意，根据 OAuth 规范，请求体 不参与签名。
- 如果在请求中包含参数，按字母顺序排列参数可以避免很多麻烦。
- 从 WooCommerce 3.0 开始支持 Authorization 头信息。

版本号 #2
由 壳壳虫 创建于 2026-03-26 06:00:56 UTC
由 壳壳虫 更新于 2026-03-26 06:00:56 UTC