附加结账字段

目录
可用的字段位置
联系信息
地址
订单信息
访问值
辅助方法
访客客户
已登录的客户
访问所有字段
直接访问值
复选框的值
支持的字段类型
使用 API
选项
常规设置
字段配置
validate_callback 函数示例。此函数将检查值是否为电子邮件地址。
text 字段的选项
select 字段的选项
options 值的示例
checkbox 字段的选项
渲染复选框字段
渲染选择字段
获得焦点之前的选择输入框
获得焦点时的选择输入框
验证和清理
清理
使用 woocommerce_sanitize_additional_field 过滤器
清理示例

验证
单个字段验证
使用 woocommerce_validate_additional_field 动作
WP_Error 对象
单个字段验证示例
多个字段验证
使用 woocommerce_blocks_validate_location_{location}_fields 动作
地址验证示例
通过 JSON Schema 实现条件可见性和验证
JSON Schema 结构
Document 对象
示例
必填且可见的字段
验证结构化数据示例
$data 关键字和 JSON 指针
不在规范中的关键字和值
评估逻辑
性能考虑
向后兼容性
响应字段保存事件
响应字段读取事件
一个完整的示例

对于开发者和商家来说，一个常见的用例是向结账表单添加新的字段，以收集有关客户或其订单的更多信息。

本文档将概述扩展程序应采取的步骤，以注册一些附加的结账字段。

可用的字段位置
附加的结账字段可以在三个不同的位置进行注册：

标题 标识符联系信息 contact 地址（配送 和 账单） address

标题	标识符
订单信息	`order`

一个字段只能显示在一个位置，在同一个注册中，无法在多个位置渲染相同的字段。

联系信息

联系信息部分当前位于表单的顶部。它包含 `email` 字段以及任何其他附加字段。

显示联系信息部分，其中渲染了两个字段：电子邮件和一个附加的结账字段（可选）

在此处渲染的字段将保存到购物者的帐户中。它们将在购物者的 “帐户详情 ”部分可见和可编辑。

地址

“地址 ”
部分当前包含一个用于配送地址和账单地址的表单。可以注册附加的结账字段，使其出现在这些表单中。

显示配送地址表单，其中在底部显示附加的结账字段

在此处注册的字段将同时保存到客户 和 订单中，因此回头客无需再次填写这些值。

如果在 `address` 位置注册了一个字段，它将同时出现在配送地址和账单地址中。
不可能将该字段仅显示在其中一个地址中。

您最终将为该字段收集两个值，一个用于配送，一个用于账单。

订单信息

作为 “额外结账字段 ”功能的一部分，结账阻止块现在包含一个名为 “订单信息阻止块 ”的新内部阻止块。

此阻止块用于渲染不属于联系信息或地址信息的部分，例如，它可能是一个 “您是通过什么途径了解我们的 ”
字段或一个 “礼品留言 ”字段。

在此处渲染的字段将保存到订单中。它们不会成为客户保存的地址或帐户信息的一部分。新订单不会预先填充
任何之前使用过的值。

包含额外结账字段的订单信息部分

默认情况下，此阻止块将作为结账表单的最后一步显示，但您可以使用古腾堡阻止控件在编辑器中移动它。

在文章编辑器中的订单信息阻止块

访问值
附加字段会分别保存到客户元数据和订单元数据中的独立元键中。您可以使用辅助方法访问它们，也可以直接
使用元键。我们建议使用辅助方法，因为它们不太可能发生变化，可以处理未来的迁移，并且将支持未来的增
强功能（例如，从不同的位置读取）。

对于地址字段，会保存两个值：一个用于配送，一个用于账单。如果客户选择了 “使用相同的地址用于账单 ” ，则这些值将相同，但仍然独立保存。

对于联系信息和订单字段，每个字段只保存一个值。

辅助方法

CheckoutFields 提供了一个函数用于访问客户和订单中的值，该函数是 get_field_from_object。

要访问客户的账单和 /或配送信息：

use Automattic\WooCommerce\Blocks\Package;
use Automattic\WooCommerce\Blocks\Domain\Services\CheckoutFields;

$field_id = 'my-plugin-namespace/my-field';
$customer = wc()->customer; // 或者 new WC_Customer($id)
$checkout_fields = Package::container()->get(CheckoutFields::class); $my_customer_billing_field = $checkout_fields->get_field_from_object($field_id, $customer, 'billing');
$my_customer_shipping_field = $checkout_fields->get_field_from_object($field_id, $customer,
' shipping');

要访问订单字段：

use Automattic\WooCommerce\Blocks\Package;
use Automattic\WooCommerce\Blocks\Domain\Services\CheckoutFields;

$field_id = 'my-plugin-namespace/my-field';
$order = wc_get_order(1234);
$checkout_fields = Package::container()->get(CheckoutFields::class); $my_order_billing_field = $checkout_fields->get_field_from_object($field_id, $order, 'billing');
$my_order_shipping_field = $checkout_fields->get_field_from_object($field_id, $order, 'shipping');

在订单提交后，保存到客户的数据和保存到订单的数据将是相同的。 客户可以更改 _未来 _ 订单的值，或者在他们的 “我的账户 ”网页中进行更改。
如果您在特定时间点查看客户数据（例如，在订单提交时），请从订单对象中访问它；如果您需要获取最新的
数据，请从客户对象中访问它。

访客客户

当访客客户提交包含附加字段的订单时，这些字段将保存到其会话中，因此只要客户仍然有有效的会话，这些
值始终可用。

已登录的客户

对于已登录的客户，值只有在他们提交订单后才会持久保存。
在订单提交生命周期中访问已登录的客户对象将返回 null 或过时的数据。

如果在订单提交钩子中执行以下操作，将返回以前的数据（而不是当前插入的数据）：

$customer = new WC_Customer($order->customer_id); // 或者 new WC_Customer(1234) $my_customer_billing_field = $checkout_fields->get_field_from_object($field_id, $customer, 'billing');

相反，如果您想执行一些额外的验证 /数据移动，请始终访问最新的数据：

$customer = wc()->customer // 这将返回具有其会话的当前客户。
$my_customer_billing_field = $checkout_fields->get_field_from_object($field_id, $customer, 'billing');

访问所有字段

您可以使用 get_all_fields_from_object 函数来访问保存到订单或客户对象上的所有附加字段。

use Automattic\WooCommerce\Blocks\Package;
use Automattic\WooCommerce\Blocks\Domain\Services\CheckoutFields;

$order = wc_get_order(1234);
$checkout_fields = Package::container()->get(CheckoutFields::class); $order_additional_billing_fields = $checkout_fields->get_all_fields_from_object($order, 'billing');
$order_additional_shipping_fields = $checkout_fields->get_all_fields_from_object($order,
' shipping');
$order_other_additional_fields = $checkout_fields->get_all_fields_from_object($order, 'other'); // 联络信息和订单信息都保存在附加组中的同一位置。

这将返回一个包含所有值的数组。 默认情况下，它只包含当前注册的字段。
如果您想包含不再注册的字段，可以传递一个第三个 `true` 参数。

```
$order = wc_get_order( 1234 );
$checkout_fields = Package::container()->get( CheckoutFields::class );
$order_additional_billing_fields = $checkout_fields->get_all_fields_from_object( $order,
'billing' ); // array( 'my-plugin-namespace/my-field' => 'my-value' );

$order_additional_billing_fields = $checkout_fields->get_all_fields_from_object( $order,
'billing', true  ); // array( 'my-plugin-namespace/my-field' => 'my-value', 'old-
namespace/old-key' => 'old-value' );
```

直接访问值

虽然不推荐，但您可以使用直接的元键来访问某些值。 这对于外部引擎或仅提供对元值访问的页面 /
电子邮件构建器非常有用。

值保存在预定义的 %prefix%下，这是为了能够查询字段，而无需知道该字段在哪个 ID 下注册。 对于键为
`'my-plugin-namespace/my-field'` 的字段，如果它是 %address%，其元键将是以下内容：

- `_wc_billing/my-plugin-namespace/my-field`
- `_wc_shipping/my-plugin-namespace/my-field`

或者，如果它是 %contact%/订单字段，则为以下内容：

- `_wc_other/my-plugin-namespace/my-field`。

这些 %prefix%是 `%CheckoutFields%` 类的一部分，可以使用以下常量访问：

```
echo ( CheckoutFields::BILLING_FIELDS_PREFIX ); // _wc_billing/
echo ( CheckoutFields::SHIPPING_FIELDS_PREFIX ); // _wc_shipping/
echo ( CheckoutFields::OTHER_FIELDS_PREFIX ); // _wc_other/
```

`%CheckoutFields%` 提供了一些辅助函数，用于根据组名或键来获取组名或键：

```
CheckoutFields::get_group_name( "_wc_billing" ); // "billing"
CheckoutFields::get_group_name( "_wc_billing/" ); // "billing"

CheckoutFields::get_group_key( "shipping" ); // "_wc_shipping/"
```

用例是构建键名以直接访问元数据：

```
$key      = CheckoutFields::get_group_key( "other" ) . 'my-plugin/is-opt-in';
$opted_in = get_user_meta( 123, $key, true ) === "1" ? true : false;
```

复选框的值

当直接访问复选框的值时，它将返回 `"1"` 表示真，`"0"` 表示假，或者如果值不存在则返回 `""`。
只有提供的函数才能将其转换为布尔值。

支持的字段类型

以下字段类型受支持：

- `select`
- `text`
- `checkbox`

计划扩展此列表，但目前这些是可用的类型。

使用 API

要注册额外的结账字段，您必须使用 `woocommerce_register_additional_checkout_field` 函数。

建议在 `woocommerce_init` 动作之后运行此函数。

注册函数接受一个选项数组，用于描述您的字段。 某些字段类型需要额外的选项。

选项

常规设置

这些选项适用于所有字段类型（除非在少数情况下，具体情况会在 “选项 ”中注明）。

字段配置

本节描述了如何配置字段。

选项名称	描述	是否必填	示例	默认值

`id`	字段的 ID。这应该为您的字段提供唯一的标识符。它由命名空间和字段名称组成，中间用 `/` 分隔。	是	`plugin-namespace/how-did-you-hear`	无默认值 - 必须提供。
`label`	在您的字段上显示的标签。这将也是占位符。	是	`How did you hear about us?`	无默认值 - 必须提供。
`optionalLabel`	如果字段是可选的，则在您的字段上显示的标签。这将也是占位符。	否	`How did you hear about us? (Optional)`	默认值将是 `label` 的值，并在其后附加 `(optional)`。
`location`	字段的渲染位置。	是	`contact`，`address`，或 `order`	无默认值 - 必须提供。
`type`	您要渲染的字段类型。默认为 `text`，并且必须与受支持的字段类型之一匹配。	否	`text`，`select`，或 `checkbox`	`text`
`attributes`	要在字段的输入元素上渲染的附加属性的数组。这 _不支持 _ `select` 字段。	否	`['data-custom-data' => 'my-custom-data' ]`	`[]`
`required`	可以是布尔值或 JSON 结构化数据数组。如果为布尔值且为 `true`，则购物者必须在结账过程中为该字段提供值。对于复选框字段，购物者必须选中复选框才能完成订单。如果为 JSON 结构化数据数组，则该字段是否必填取决于结构化数据的条件。请参阅通过 JSON 结构化数据实现条件可见性和验证。	否	`true` 或 `["type" => "object", "properties" => [...]]`	`false`
`hidden`	可以是布尔值或 JSON 结构化数据数组。如果用作布尔值，则必须为 `false`。如果为 JSON 结构化数据数组，则该字段是否隐藏取决于结构化数据的条件。请参阅通过 JSON 结构化数据实现条件可见性和验证。	否	`false` 或 `["type" => "object", "properties" => [...]]`	`false`

`validation`	定义字段验证规则的 JSON 结构化数据对象的数组。请参阅通过 JSON 结构化数据实现条件可见性和验证。	否	`[{"type": "object", "properties": {...}}]`	`[]`
`sanitize_callback`	在发布时，用于清理客户提供的值的函数。	否	参见下面的示例	默认情况下，字段的值会不作更改地返回。
`validate_callback`	在发布时，用于验证客户提供的值的函数。这在清理之后运行。	否	参见下面的示例	默认验证函数将在字段为必填但没有值时，向响应添加一个错误。请参阅默认验证函数。

````
##### `sanitize_callback` 函数示例。此函数将从值中删除空格。<!-- omit from toc -->

```php
'sanitize_callback' => function( $field_value ) {
	return str_replace( ' ', '', $field_value );
},
````

`validate_callback` 函数示例。此函数将检查值是否为电子邮件地址。

```
'validate_callback' => function( $field_value ) {
	if ( ! is_email( $field_value ) ) {
		return new WP_Error( 'invalid_alt_email', '请确保您的备用电子邮件地址符合正确的格式。' );
	}
},
```

`text` 字段的选项

`text` 字段除了上面列出的常规设置之外，没有其他额外的选项。

`select` 字段的选项

除了上面列出的选项之外，`select` 字段还必须使用 `options` 选项进行注册。这用于指定用户可以从中选择的选项。

`select` 字段默认情况下不会选择任何值；如果该字段是必填的，则用户必须选择一个值。

您可以通过在注册字段时传递 `placeholder` 值，来设置在 `select` 字段中显示的占位符。这将是 `select` 中的第一个选项，并且如果该字段是必填的，则用户无法选择此选项。

选项名称	描述	必需?	示例	默认值
```

`options`	用于在 `select` 输入框中显示的选项数组。每个选项必须是一个包含 `label` 和 `value` 属性的数组。每个条目必须具有唯一的 `value` 。任何重复的选项将被删除。 `value` 是在结账过程中提交到服务器的值，而 `label` 只是此值的用户友好表示，它不会以任何方式传输到服务器。	是	参见下文	无默认值 - 必须提供。
`placeholder`	如果设置了此值，则用户将在 `select` 中看到此选项。如果 `select` 是必填的，则用户无法选择此选项。	否	选择一个角色	Select a $label

`options` 值的示例

```
[

□[
□□'value' => 'store_1',
□□'label' =>我们的伦敦店'
□],
□[
□□'value' => 'store_2',
□□'label' =>我们的巴黎店'
□],
□[
□□'value' => 'store_3',
□□'label' =>我们的纽约店'
□]
]
```

`checkbox` 字段的选项

除了上面列出的选项之外， `checkbox` 字段还支持在必填时，如果未选中，则显示错误信息。

选项名称	描述	必需?	示例	默认值
`error_message`	如果复选框未选中，则显示的自定义消息。	否	您必须确认您已满 18 岁才能下单。	如果希望继续，请选中此框。

属性

支持为复选框和文本字段添加额外的属性。但是，**目前无法**将属性添加到选择字段。

这些属性与 `input` 元素的 HTML 属性一一对应（`pattern` 属性除外，复选框不支持）。

支持的属性包括：

- `data-*` 属性
- `aria-*` 属性
- `autocomplete`
- `autocapitalize`
- `pattern` (复选框字段不支持)
- `title`
- `maxLength` (等同于 HTML 属性 `maxlength`)
- `readOnly` (等同于 HTML 属性 `readonly`)

`maxLength` 和 `readOnly` 使用驼峰命名法，因为这些属性会渲染到 React 元素上，必须以这种格式接收。

某些属性有意不传递给字段，这些是 `autofocus` 和 `disabled`。我们欢迎接收反馈，并在提供有效的用例时调整此行为。

使用示例

渲染文本字段

此示例演示如何在地址部分渲染文本字段：

```php
add_action(
	'woocommerce_init',
	function() {
		woocommerce_register_additional_checkout_field(
			array(
				'id'            => 'namespace/gov-id',
				'label'         => 'Government ID',
				'optionalLabel' => 'Government ID (optional)',
				'location'      => 'address',
				'required'      => true,
				'attributes'    => array(
```

```
□□□□□'autocomplete'     => 'government-id',
□□□□□'aria-describedby' => 'some-element',
□□□□□'aria-label'       => 'custom aria label',
□□□□□'pattern'          => '[A-Z0-9]{5}', □□写5□母和数字组成的字符串。
□□□□□e'            => 'Title to show on hover',
□□□□□'data-custom'      => 'custom data',
□□□□),
□□□),
□□);
□}
);
```

这将在地址表单中生成以下内容（账单表单与此相同）：

显示在地址表单底部的 Government ID 字段

生成的 HTML 标记如下：

```
□<input type="text" id="shipping-namespace-gov-id" autocapitalize="off"
       autocomplete="government-id" aria-label="custom aria label"
       aria-describedby="some-element" required="" aria-invalid="true"
       title="Title to show on hover" pattern="[A-Z0-9]{5}"
       data-custom="custom data" value="">
```

渲染复选框字段

这个示例演示了如何在联系信息部分渲染一个复选框字段：

```
add_action(
□'woocommerce_init',
□function() {
□□woocommerce_register_additional_checkout_field(
□□□array(
□□□□'id'       => 'namespace/marketing-opt-in',
□□□□'label'    您□否想订阅我们的新闻通讯？',
□□□□'location' => 'contact',
□□□□'type'     => 'checkbox',
□□□)
□□);
□}
```

```
);
```

这将在联系信息部分显示以下内容：

联系信息部分，其中包含一个订阅新闻通讯的复选框

请注意，由于没有提供 `optionalLabel`，因此字符串 `(optional)` 会附加到标签中。要删除该字符串，应提供 `optionalLabel` 属性以覆盖此行为。

渲染选择字段

这个示例演示了如何在订单信息部分渲染一个选择字段：

```
add_action(
	'woocommerce_init',
	function() {
		woocommerce_register_additional_checkout_field(
			array(
				'id'          => 'namespace/how-did-you-hear-about-us',
				'label'       => '您是通过什么途径了解我们的？',
				'placeholder' => '选择一个来源',
				'location'    => 'order',
				'type'        => 'select',
				'options'     => [
					[
						'value' => 'google',
						'label' => 'Google'
					],
					[
						'value' => 'facebook',
						'label' => 'Facebook'
					],
					[
						'value' => 'friend',
						'label' => '朋友推荐'
					],
					[
						'value' => 'other',
						'label' => '其他 => '
					],
				]
```

```
□□□)
□□);
□}
);
```

这将在订单信息部分以如下方式进行渲染：

获得焦点之前的选择输入框

获得焦点之前的选择输入框

获得焦点时的选择输入框

获得焦点时的选择输入框

如果不想强制用户选择一个值，可以通过将 `required` 选项设置为 `false`，将选择框标记为可选。

验证和清理

可以使用 WordPress 动作钩子，为附加的结账字段添加自定义的验证和清理功能。

这些操作发生在两个地方：

1. 在结账过程中更新和提交表单，以及
2. 在“我的账户”区域更新地址/联系信息。

清理

清理用于确保字段的值具有特定的格式。例如，在获取政府身份证件时，您可能希望将其格式化为所有字母大写且没有空格。此时，不应检查该值的有效性。这将在稍后进行。此步骤仅用于为验证做好准备。

使用 `woocommerce_sanitize_additional_field` 过滤器

为了运行自定义的字段清理函数，您可以在注册时使用 `sanitize_callback` 函数，或者使用 `woocommerce_sanitize_additional_field` 过滤器。

参数	类型	描述
`$field_value`	`boolean|string`	字段的值。

参数	类型	描述
`$field_key`	`string`	字段的 ID。这与字段注册时使用的 ID 相同。

清理示例

此示例展示了如何移除空格并将所有字母转换为大写，以清理我们在上面添加的“政府 ID”示例字段。

```
add_action(
	'woocommerce_sanitize_additional_field',
	function ( $field_value, $field_key ) {
		if ( 'namespace/gov-id' === $field_key ) {
			$field_value = str_replace( ' ', '', $field_value );
			$field_value = strtoupper( $field_value );
		}
		return $field_value;
	},
	10,
	2
);
```

验证

在附加的结账字段系统中，有两个验证阶段。第一个阶段是根据字段的键和值验证单个字段。

单个字段验证

使用 `woocommerce_validate_additional_field` 动作

当 `woocommerce_validate_additional_field` 动作被触发时，回调函数会接收到字段的键、字段的值以及一个 `WP_Error` 对象。

要将验证错误添加到响应中，请使用 `WP_Error::add` 方法。

参数	类型	描述

`WP_Error` 对象

当您向 `WP_Error` 对象添加错误时，该错误应该具有唯一的错误代码。
您可能希望在错误代码前添加插件命名空间，以减少冲突的可能性。
使用其他插件中已使用的代码可能会导致错误信息被覆盖或显示在不同的位置。

单个字段验证示例

以下示例展示了如何将自定义验证应用于上述 `namespace/gov-id` 文本字段。 此处的代码确保该字段由 5
个字符组成，可以是大写字母或数字。 上述示例中的 sanitization
函数确保删除所有空格并将所有字母大写，因此此检查是额外的安全措施，以确保输入与模式匹配。

```
add_action(
'woocommerce_validate_additional_field',
function ( WP_Error $errors, $field_key, $field_value ) {
if ( 'namespace/gov-id' === $field_key ) {
$match = preg_match( '/[A-Z0-9]{5}/', $field_value );
if ( 0 === $match || false === $match ) {
$errors->add( 'invalid_gov_id请保您的政府 ID 符合正确的格式。' );
}
}
},
10,
3
);
```

重要的是要注意，此操作必须向它接收到的 `WP_Error` 对象 添加 错误。 返回一个新的 `WP_Error`
对象或任何其他值会导致错误不显示。

如果没有遇到任何验证错误，该函数可以直接返回 void。

多个字段验证

在某些情况下，一个字段的有效性取决于另一个字段的值，例如，根据购物者所在的国家 /地区来验证政府 ID
的格式。 在这种情况下，仅验证单个字段（如上所示）是不够的，因为在
`woocommerce_validate_additional_field` 操作期间，国家 /地区可能未知。

为了解决这个问题，可以在字段呈现的上下文环境中验证该字段。 位于该位置的其他字段将传递给此操作。

使用 `woocommerce_blocks_validate_location_{location}_fields` 动作

此动作将为每个可以渲染附加字段的位置（ `address`、`contact` 和 `order`）触发一次。对于 `address`
，它会触发两次，一次是用于账单地址，另一次是用于配送地址。

回调函数接收到相同位置的其他附加字段的键和值。

需要注意的是，在其他位置渲染的任何字段都不会传递给此动作，但是可以通过访问客户或订单对象来获取这
些值，但这不受支持，并且未来版本不保证兼容性。

参数	类型	描述

`$errors`	`WP_Error`	一个错误对象，包含在处理请求时已经遇到的错误。如果尚未添加任何错误，它仍然是一个 `WP_Error` 对象，但为空。
`$fields`	`array`	在此位置渲染的字段。
`$group`	`'billing'|'shipping'|'other'`	如果该动作是针对地址位置，则此处会设置地址类型。如果是针对联系方式或订单，则此值为 'other'。

这些钩子在以下几个地方会被触发。

- 在使用结账块或 Store API 进行结账时：
  - `woocommerce_blocks_validate_location_address_fields` (x2)
  - `woocommerce_blocks_validate_location_contact_fields`
  - `woocommerce_blocks_validate_location_other_fields`
- 在“我的账户”区域更新地址时：
  - `woocommerce_blocks_validate_location_address_fields` (**x1** - 仅针对正在编辑的地址)
- 在“我的账户”区域更新“账户详情”部分时：
  - `woocommerce_blocks_validate_location_contact_fields`

地址验证示例

在这个示例中，假设除了 `namespace/gov-id` 之外，还注册了一个名为 `namespace/confirm-gov-id` 的字段。这个字段将用于确认政府 ID 字段。

下面的示例说明了如何验证确认字段的值是否与主字段的值匹配。

```
add_action(
	'woocommerce_blocks_validate_location_address_fields',
	function ( \WP_Error $errors, $fields, $group ) {
		if ( $fields['namespace/gov-id'] !== $fields['namespace/confirm-gov-id'] ) {
			$errors->add( 'gov_id_mismatch', '确保您的政府 ID 与确认信息一致。' );
		}
	},
	10,
	3
);
```

如果这些字段显示在"联系方式"位置，则代码相同，只是使用的钩子不同：`woocommerce_blocks_validate_location_contact_fields`。

通过 JSON Schema

实现条件可见性和验证
required、 hidden 和 validation 属性接受一个 JSON Schema 数组，用于为字段创建条件逻辑。
这允许您根据其他字段的值动态控制字段的可见性、必填状态和验证规则。

Schema 在前端实时评估，并在后端进行任何更新时进行评估。 这确保了快速响应的 UI ，以及客户端和服务器之间的一致结果。

JSON Schema 结构
数组中的每个 Schema 应该是一个有效的 JSON Schema 对象，该对象定义了何时应应用该属性的条件。 Schema 会针对当前购物车和结账状态进行评估，其中包括所有字段值以及各种选项（支付、配送、客户）。

JSON Schema 对象的基本结构如下：

{
"type": "object",
"properties": {
"fieldId": {
"enum": ["value1", "value2"]
}
},
"required": ["fieldId"]
}

如果您不熟悉 JSON Schema，您可以从 官方网站 了解其基本知识，或者参考一些使用的库，例如 AJV或 OPIS.。 Checkout 构建了一个基于这两个库的抽象层。

Document 对象

当您撰写规则时，您实际上是在撰写一个 document 对象的局部 结构化数据
，本质上是描述您希望字段处于何种理想状态，例如是否需要或隐藏。

重要： document 对象中的所有 属性 都使用 snake_case 命名约定（例如， total_price、 shipping_rates 、 customer_note），而不是 camelCase。

document 对象的一个示例如下：

Document


```
{
"cart": {
"coupons": [
"my_coupon"
],
"shipping_rates": [
"free_shipping:1"
],
"items": [
27,
27,
68
],
"items_type": [
"simple",
"
],
"items_count": 3,
"items_weight": 0,
"needs_shipping": true,
"prefers_collection": false,
"totals": {
"total_price": 6600,
"total_tax": 600
},
"extensions": {}
},
"checkout": {
"create_account": false,
"customer_note": "",
"additional_fields": {
"namespace/myorder-field": "myvalue"
},
"payment_method": "bacs"
},
"customer": {
"id": 1,
```

```
  "billing_address": {
    "first_name": "",
    "last_name": "",
    "company": "",
    "address_1": "1",
    "address_2": "2",
    "city": "",
    "state": "",
    "postcode": "08000",
    "country": "US",
    "email": "email@example.com",
    "phone": "1234567890",
    "namespace/myfield": "myvalue"
  },
  "shipping_address": {
    "first_name": "",
    "last_name": "",
    "company": "",
    "address_1": "1",
    "address_2": "2",
    "city": "",
    "state": "",
    "postcode": "08000",
    "country": "US",
    "phone": "1234567890",
    "namespace/myfield": "myvalue"
  },
  "additional_fields": {
    "namespace/mycontact-field": "myvalue"
  },
  "address": {
    "first_name": "",
    "last_name": "",
    "company": "",
    "address_1": "1",
    "address_2": "2",
    "city": "",
    "state": "",
    "postcode": "08000",
```

```
    "country": "US",
    "phone": "1234567890",
    "namespace/myfield": "myvalue"
  }
}
}
```

它完整的结构是这样的:

文档结构

```
{
"$schema": "http://json-schema.org/draft-07/schema#",
"title": 购物车和结账文档对象结构",
"description": "物车、结账和客户信息的文档对象结构，用于条件可见性、字段要求和验证。",
"type": "object",
"properties": {
  "cart": {
    "type": "object",
    "description"关于购物车的的信息",
    "properties": {
      "coupons": {
        "type": "array",
        "descriptio应用于购物车的优惠券代码列表",
        "items": {
          "type": "string"
        }
      },
      "shipping_rates": {
        "type": "array",
        "descriptio当前选择的运输费率列表",
        "items": {
          "type": "string",
          "descriptio用完整运输费率 ID 的运输费率标识符，例如: method_id:instance_id，例如: flat_rate:1"
        }
      },
      "items": {
        "type": "array",
        "descriptio购物车中产品 ID 的列表，ID 将根据产品在购物车中的数量进行重复，如果购物车中有 2 个产品 ID 为 1 的产品，则数组将包含 2 个产品
```

```
ID 1 的条目。仅支持整数数量，不支持浮点数（四舍五入到最接近的整数）。",
□□□□□"items": {
□□□□□□"type": "integer"
□□□□□}
□□□□},
□□□□"items_type": {
□□□□□"type": "array",
□□□□□"description":"购物□中商品类型，例如: simple、variation、subscription 等。",
□□□□□"items": {
□□□□□□"type": "string"
□□□□□}
□□□□},
□□□□"items_count": {
□□□□□"type": "integer",
□□□□□"description":"购物□中商品的总数量",
□□□□□"minimum": 0
□□□□},
□□□□"items_weight": {
□□□□□"type": "number",
□□□□□"description":"购物□中商品的总重量",
□□□□□"minimum": 0
□□□□},
□□□□"needs_shipping": {
□□□□□"type": "boolean",
□□□□□"description":"购物□中的商品是否需要运输"
□□□□},
□□□□"prefers_collection": {
□□□□□"type": "boolean",
□□□□□"description":"用户□否更喜欢使用本地自提"
□□□□},
□□□□"totals": {
□□□□□"type": "object",
□□□□□"description":"购物□总价信息",
□□□□□"properties": {
□□□□□□"total_price": {
□□□□□□□"type": "integer",
□□□□□□□"description":"购物□总□，以最小货币单位表示（例如：分），在应用所有折扣、运费和税费后"
□□□□□□},
□□□□□□"total_tax": {
```

```
        "type": "integer",
        "description": "...",
      }
    },
    "additionalProperties": false
  },
  "extensions": {
    "type": "object",
    "description": "... Store API ... extensions ...",
  }
},
"additionalProperties": false
},
"checkout": {
"type": "object",
"description": "...",
"properties": {
  "create_account": {
    "type": "boolean",
    "description": "... false ...",
  },
  "customer_note": {
    "type": "string",
    "description": "...",
  },
  "additional_fields": {
    "type": "object",
    "description": "... 'order' ...",
    "additionalProperties": {
      "type": "string"
    },
    "patternProperties": {
      "^[a-zA-Z0-9_-]+/[a-zA-Z0-9_-]+$": {
        "type": "string",
        "description": "..."
      }
    }
  },
  "payment_method": {
```

```
        "type": "string",
        "description": "ID"
      }
    },
    "additionalProperties": false
  },
  "customer": {
    "type": "object",
    "description": "",
    "properties": {
      "id": {
        "type": "integer",
        "description": "ID 0"
      },
      "billing_address": {
        "$ref": "#/definitions/address",
        "description": ""
      },
      "shipping_address": {
        "$ref": "#/definitions/address",
        "description": ""
      },
      "additional_fields": {
        "type": "object",
        "description": "'contact' ",
        "additionalProperties": {
          "type": "string"
        },
        "patternProperties": {
          "^[a-zA-Z0-9_-]+/[a-zA-Z0-9_-]+$": {
            "type": "string",
            "description": ""
          }
        }
      },
      "address": {
        "$ref": "#/definitions/address",
        "description": ""
      }
```

```
    },
    "additionalProperties": false
  }
},
"additionalProperties": false,
"definitions": {
  "address": {
    "type": "object",
    "description": "",
    "properties": {
      "first_name": {
        "type": "string",
        "description": ""
      },
      "last_name": {
        "type": "string",
        "description": ""
      },
      "company": {
        "type": "string",
        "description": ""
      },
      "address_1": {
        "type": "string",
        "description": ""
      },
      "address_2": {
        "type": "string",
        "description": ""
      },
      "city": {
        "type": "string",
        "description": ""
      },
      "state": {
        "type": "string",
        "description": "CA、TX、NY London、"
      },
      "postcode": {
```

```
□□□□□"type": "string",
□□□□□"descriptio邮□编码"
□□□□},
□□□□"country": {
□□□□□"type": "string",
□□□□□"descriptio国□代码□例如: US、UK）"
□□□□},
□□□□"email": {
□□□□□"type": "string",
□□□□□"format": "email",
□□□□□"description": "E□□il
□□□□},
□□□□"phone": {
□□□□□"type": "string",
□□□□□"descriptio电□号码"
□□□□}
□□□},
□□□"additionalProperties": {
□□□□"type": "string",
□□□□"description□有 "address' 位置的附加字段将在此处以地址对象属性的形式出现"
□□□},
□□□"patternProperties": {
□□□□"^[a-zA-Z0-9_-]+/[a-zA-Z0-9_-]+$": {
□□□□□"type": "string",
□□□□□"descriptio使□命名□间标识符（例如: 'namespace/field-name'）的具有 'address' 位置的附加字段"
□□□□}
□□□}
□□}
□}
}
```

```
# 关于 %s 的技术文档

本文档提供了关于 %s 的详细信息。

## 安装

1. 下载 %s 的最新版本。
2. 解压下载的文件。
```

3. 按照安装说明进行操作。

配置

%s 的配置可以通过修改配置文件进行。配置文件位于 `%s/config/config.ini`。

以下是一些常用的配置选项：

* `server_address`：服务器地址。
* `server_port`：服务器端口。
* `database_host`：数据库主机。
* `database_name`：数据库名称。

使用

要使用 %s，请按照以下步骤操作：

1. 启动 %s 服务器。
2. 使用客户端连接到服务器。
3. 执行所需的命令。

示例

以下是一个使用 %s 的示例：

%s --server-address %s --server-port %d

故障排除

如果在使用 %s 时遇到问题，请参考以下故障排除步骤：

* 检查配置文件是否正确。
* 检查服务器是否正在运行。
* 检查客户端是否已正确配置。

详情

有关 %s 的更多信息，请访问 https://www.example.com/%s。

启用高级功能

要启用高级功能，请修改配置文件并设置 `enable_advanced_features` 为 `true`。

日志

%s 将日志信息写入文件 `%s/logs/application.log`。

许可证

%s 采用 Apache 2.0 许可证。

联系方式

如有任何问题，请通过电子邮件发送至 `support@example.com`。

更新日志

* **2023-03-08**: 修复了 %s 的一个 bug。
* **2023-02-15**: 增加了对 %s 的新功能支持。
* **2023-01-01**: 发布了 %s 的第一个版本。

常见问题解答

* **问：%s 是什么？**
* 答：%s 是一个用于 %s 的工具。

* **问：如何安装 %s？**
* 答：请参考 "安装" 部分的说明。

* **问：如何配置 %s？**
* 答：请参考 "配置" 部分的说明。

* **问：如何使用 %s？**
* 答：请参考 "使用" 部分的说明。

* **问：%s 的许可证是什么？**

```
    *    答: %s 采用 Apache 2.0 许可证。

## 贡献

欢迎您为 %s 做出贡献。请访问我们的 GitHub 仓库:
[https://github.com/example/%s](https://github.com/example/%s)。

## 感谢

感谢所有为 %s 做出贡献的人。

## 术语表

*   **Details**: 详情
*   **Enable**: 启用
*   **Mar**: 三月
*   **In**: 在
*   **Down**: 向下
```

示例

必填且可见的字段

在这个示例中，我们使字段成为必填项，并且仅当选择 “本地自提 ”时才可见。

```
'required' => [
    "type" => "object",
□"properties" => [
□□"cart" => [
□□□"properties" => [
□□□□"prefers_collection" => [
□□□□□"const" => true
□□□□]
□□□]
□□]
□]
],
'hidden' => [
□"type" => "object",
```

```
"properties" => [
  "cart" => [
    "properties" => [
      "prefers_collection" => [
        "const" => false
      ]
    ]
  ]
]
```

请注意，对于隐藏的字段，我们反转了该字段的显示状态，这意味着该字段应该仅在 `prefers_collection` 为 `false` 时才被隐藏，而 `false` 几乎适用于所有情况，除非选择了“本地自提”。

在上面的示例中，我们使用了 关键字 `const`。

验证结构化数据示例

验证与条件可见性和必填性略有不同。

在验证中，您将传入结构化数据的子集（仅适用于您的字段），其作用是验证该字段，并在出现错误时显示错误信息。

在这个示例中，我们确保 VAT 由国家代码和 8-12 个数字组成。

```
'validation' => [
  "type" => "string",
  "pattern" => "^[A-Z]{2}[0-9]{8,12}$",
  "errorMessage" => "填写有效的 VAT 代码，该代码应包含 2 个字母作为国家代码，以及 8-12 个数字。"
]
```

验证还可以针对其他字段，例如，一个备用电子邮件字段，该字段不应与当前的电子邮件匹配：

```
'validation' => [
  "type" => "string",
  "format" => "email",
  "not" => [
    "const" => [ '$data' => "/customer/billing_address/email" ]
  ],
  "errorMessage" => "填写有效的备用电子邮件。"
]
```

在上面的示例中，我们使用了 格式关键字 和 $data 来引用当前字段的字段值，方法是通过 JSON 指针 。我们还使用了 errorMessage 属性来提供自定义的错误消息。

$data 关键字和 JSON 指针

$data 关键字是在 JSON

结构化数据中引用另一个字段值的机制。在上面的示例中，我们使用它来引用账单电子邮件，通过 JSON

指针 。

在使用 JSON 指针时，需要注意以下几点：

正斜杠 / 用于在 JSON 对象中导航，因此，对于其他字段，名为 my-plugin-namespace/my-
field 的字段需要引用为 my-plugin-namespace~1my-field。
JSON
指针的导航可以从当前字段向后，也可以从根节点开始。如果您有一个地址字段，并且想要验证
电话字段，这意味着您将验证两个值，一个用于配送，一个用于账单，因此您可以以两种方式引
用电话字段：
0/customer/address/phone，它使用根导航（通过 0/ 前缀），并使用动态的 address
组，该组会根据正在验证的配送或账单值而变化。
1/phone，它使用相对指针向后移动，在这种情况下，它将访问其兄弟字段，即 phone
字段。增加数字可以向后移动更远，例如， 2/id 将访问客户 ID。

不在规范中的关键字和值

我们支持 JSON 结构化数据 Draft-07
，它比较简单，不支持所有最新的规范中的关键字和值，但我们认为它涵盖了大多数用例。此外，我们引入
了一些非标准的关键字和值，这些值不在规范中，它们的实现可能在 Opis 和 AJV (或任何未来的实现)
之间有所不同。以下是此类关键字和值的列表：

errorMessage: 用于验证的自定义错误信息。在 AJV 中，它是 errorMessage，在 Opis
中，它是 $error。我们仅支持 errorMessage，并在内部将其映射到 Opis
。目前，我们不支持在 errorMessage 中使用模板。

$data: 通过 JSON 指针 引用当前字段值。 Opis 和 AJV 使用相同的实现。

评估逻辑

对于 required:
如果数组中的任何一个结构化数据与当前的结账状态匹配，则该字段将是必填项。
对于 hidden: 如果数组中的任何一个结构化数据与当前的结账状态匹配，则该字段将被隐藏。
对于 validation:
字段的值将根据提供的部分结构化数据进行评估，如果值不匹配，则会显示错误。

性能考虑

复杂的 JSON
结构化数据条件可能会影响结账性能。请尽可能简化您的结构化数据，并限制条件的数量，仅限于您的用例
所需的数量。

向后兼容性

由于技术原因，目前还无法为字段指定元键，因为我们希望它们具有前缀并由系统进行管理。
使用短代码在结账页面上存在的字段的插件可以保持兼容性，并通过钩子来响应字段的读取和保存操作。

假设有两个字段，分别命名为 `my-plugin-namespace/address-field`（位于地址步骤 ）和 `my-plugin-namespace/my-other-field`（位于订单步骤 ），您可以：

响应字段保存事件

您可以通过钩入 `woocommerce_set_additional_field_value` 动作来响应这些字段的保存事件。

```
add_action(
	'woocommerce_set_additional_field_value',
	function ( $key, $value, $group, $wc_object ) {
		if ( 'my-plugin-namespace/address-field' !== $key ) {
			return;
		}

		if ( 'billing' === $group ) {
			$my_plugin_address_key = 'existing_billing_address_field_key';
		} else {
			$my_plugin_address_key = 'existing_shipping_address_field_key';
		}

		$wc_object->update_meta_data( $my_plugin_address_key, $value, true );
	},
	10,
	4
);

add_action(
```

```
□'woocommerce_set_additional_field_value',
□function ( $key, $value, $group, $wc_object ) {
□□if ( 'my-plugin-namespace/my-other-field' !== $key ) {
□□□return;
□□}

□□$my_plugin_key = 'existing_order_field_key';

□□$wc_object->update_meta_data( $my_plugin_key, $value, true );
□},
□10,
□4
);
```

这样，您可以确保现有的系统能够继续工作，并且您的集成也能继续工作。

但是，理想情况下，您应该迁移现有的数据和系统，以便使用新的元字段。

响应字段读取事件

您可以使用 `woocommerce_get_default_value_for_{$key}` 过滤器来提供不同的默认值（例如，来自另一个元字段的值）：

```
add_filter(
□"woocommerce_get_default_value_for_my-plugin-namespace/address-field",
□function ( $value, $group, $wc_object ) {

□□if ( 'billing' === $group ) {
□□□$my_plugin_key = 'existing_billing_address_field_key';
□□} else {
□□□$my_plugin_key = 'existing_shipping_address_field_key';
□□}

□□return $wc_object->get_meta( $my_plugin_key );
□},
□10,
□3
);

add_filter(
```

```
□"woocommerce_get_default_value_for_my-plugin-namespace/my-other-field",
□function ( $value, $group, $wc_object ) {

□□$my_plugin_key = 'existing_order_field_key';

□□return $wc_object->get_meta( $my_plugin_key );
□},
□10,
□3
);
```

一个完整的示例

在这个完整的示例中，我们将注册 “政府身份证 ”文本字段，并验证它是否符合特定的格式。

这个示例只是前面共享的示例的组合版本。

```
add_action(
□'woocommerce_init',
□function() {
□□woocommerce_register_additional_checkout_field(
□□□array(
□□□□'id'            => 'namespace/gov-id',
□□□□'label'         => 'Government ID',
□□□□'location'      => 'address',
□□□□'required'      => true,
□□□□'attributes'    => array(
□□□□□'autocomplete' => 'government-id',
□□□□□'pattern'      => '[A-Z0-9]{5}'□大□□母和数字组成的 5 个字符的字符串。
□□□□□e'        => 'Your 5-digit Government ID',
□□□□),
□□□),
□□);
□□woocommerce_register_additional_checkout_field(
□□□array(
□□□□'id'            => 'namespace/confirm-gov-id',
□□□□'label'         => 'Confirm government ID',
□□□□'location'      => 'address',
```

```
        'required'      => true,
        'attributes'    => array(
            'autocomplete' => 'government-id',
            'pattern'      => '[A-Z0-9]{5}',
            'title'        => 'Confirm your 5-digit Government ID',
        ),
    ),
);

add_action(
    'woocommerce_sanitize_additional_field',
    function ( $field_value, $field_key ) {
        if ( 'namespace/gov-id' === $field_key || 'namespace/confirm-gov-id' === $field_key ) {
            $field_value = str_replace( ' ', '', $field_value );
            $field_value = strtoupper( $field_value );
        }
        return $field_value;
    },
    10,
    2
);

add_action(
    'woocommerce_validate_additional_field',
    function ( WP_Error $errors, $field_key, $field_value ) {
        if ( 'namespace/gov-id' === $field_key ) {
            $match = preg_match( '/[A-Z0-9]{5}/', $field_value );
            if ( 0 === $match || false === $match ) {
                $errors->add( 'invalid_gov_id', '                                    ' );
            }
        }
        return $errors;
    },
    10,
    3
);
}
);
```

```
add_action(
□'woocommerce_blocks_validate_location_address_fields',
□function ( \WP_Error $errors, $fields, $group ) {
□□if ( $fields['namespace/gov-id'] !== $fields['namespace/confirm-gov-id'] ) {
□□□$errors->add( 'gov_id_mismatch'□□□□□□□□□□□□□□□□□□□□□□□□□□□' );
□□}
□},
□10,
□3
);
```

□□□ #2
□ □□□ □□□ 2026-03-09 11:53:58 UTC
□ □□□ □□□ 2026-03-09 11:53:58 UTC