处理脚本、样式和数据

目录
问题
解决方案
IntegrationInterface 方法
get_name()
initialize()
get_script_handles()
get_editor_script_handles()
get_script_data()
使用示例
获取 get_script_data 中添加的数据

问题
您是一位扩展开发者，为了允许用户在客户端与您的扩展进行交互，您编写了一些 CSS 和 JavaScript 代码，希望这些代码能够包含在网页中。 您的 JavaScript
代码还依赖于一些服务器端数据，并且您希望这些数据能够被您的脚本访问。

解决方案
您可以使用 IntegrationRegistry 来注册一个 IntegrationInterface。
这将是一个类，用于处理脚本、样式和数据的加载。
您可以为每个块（迷你购物车、购物车和结账）使用不同的 IntegrationInterface
，也可以使用同一个，这完全取决于您的用例。

您应该使用以下钩子： woocommerce_blocks_mini-cart_block_registration、 woocommerce_blocks_cart_block_registration 和 woocommerce_blocks_checkout_block_registration。这些钩子会将 IntegrationRegistry 的实例传递给回调函数。

然后，您可以使用此对象的 register 方法来注册您的 IntegrationInterface。

IntegrationInterface 方法
首先，我们需要创建一个集成类，即我们的 IntegrationInterface。 这将是一个实现 WooCommerce Blocks 接口的类，该接口名为 IntegrationInterface。
在本部分中，我们将逐步介绍接口的成员，并讨论它们的用途。

get_name()

这是 IntegrationInterface 为您的集成提供命名空间的方式。 在这里选择的名称应与您的扩展唯一。
此方法应返回一个字符串。

initialize()

您应该在此处放置任何设置或初始化的代码。 例如，您可以在此处注册您的扩展所需的脚本和样式。
此方法不应返回任何值。

get_script_handles()

在此处放置您希望在前端客户端加载的任何脚本的句柄。 此方法应返回一个字符串数组。

get_editor_script_handles()

在此处放置您希望在编辑器客户端加载的任何脚本的句柄。 此方法应返回一个字符串数组。

get_script_data()

您可以在此处设置希望在前端脚本中可用的值。 此方法应返回一个关联数组，其键将用于通过 JavaScript 函数 getSetting 获取数据。
此数组的键和值都应该是可序列化的。

使用示例

□□□□□□□□□□□ `WooCommerce Example Plugin`□□□□□□ Mini-Cart□ Cart □ Checkout □□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□□

□□□□□□□□□□□□□□□□□□□□ `/build/index.asset.php` □□□□□□□□ `DependencyExtractionWebpackPlugin` □□□□□□□□ PHP □□□□□□□□□□□□□□□□□□□□□□□□□□□□ `wp_register_script` □ `dependencies` □□□□

□□□□□□□□ `IntegrationInterface`□

```
<?php
use Automattic\WooCommerce\Blocks\Integrations\IntegrationInterface;

/**
 * Class for integrating with WooCommerce Blocks
 */
class WooCommerce_Example_Plugin_Integration implements IntegrationInterface {
□/**
□ * The name of the integration.
□ *
□ * @return string
□ */
□public function get_name() {
□□return 'woocommerce-example-plugin';
□}

□/**
□ * When called invokes any initialization/setup for the integration.
□ */
□public function initialize() {
□□$script_path = '/build/index.js';
□□$style_path = '/build/style-index.css';

    /**
     * The assets linked below should be a path to a file, for the sake of brevity
     * we will assume \WooCommerce_Example_Plugin_Assets::$plugin_file is a valid file path
    */
□□$script_url = plugins_url( $script_path, \WooCommerce_Example_Plugin_Assets::$plugin_file );
□□$style_url = plugins_url( $style_path, \WooCommerce_Example_Plugin_Assets::$plugin_file );
```

```
□□$script_asset_path = dirname( \WooCommerce_Example_Plugin_Assets::$plugin_file ) .
'/build/index.asset.php';
□□$script_asset      = file_exists( $script_asset_path )
□□□? require $script_asset_path
□□□: array(
□□□□'dependencies' => array(),
□□□□'version'      => $this->get_file_version( $script_path ),
□□□);

□□wp_enqueue_style(
□□□'wc-blocks-integration',
□□□$style_url,
□□□[],
□□□$this->get_file_version( $style_path )
□□);

□□wp_register_script(
□□□'wc-blocks-integration',
□□□$script_url,
□□□$script_asset['dependencies'],
□□□$script_asset['version'],
□□□true
□□);
□□wp_set_script_translations(
□□□'wc-blocks-integration',
□□□'woocommerce-example-plugin',
□□□dirname( \WooCommerce_Example_Plugin_Assets::$plugin_file ) . '/languages'
□□);
□}

□/**
□ * Returns an array of script handles to enqueue in the frontend context.
□ *
□ * @return string[]
□ */
□public function get_script_handles() {
□□return array( 'wc-blocks-integration' );
□}

□/**
```

```
 * Returns an array of script handles to enqueue in the editor context.
 *
 * @return string[]
 */
public function get_editor_script_handles() {
	return array( 'wc-blocks-integration' );
}
```

/** * 一个包含键值对的数据数组，这些数据在客户端提供给块。 * * @return array */ public function get_script_data() { $woocommerce_example_plugin_data = some_expensive_serverside_function(); return ['expensive_data_calculation' => $woocommerce_example_plugin_data]; }

```
/**
 * 获取文件修改时间，作为缓存清除标志，如果我们在开发模式下。
 *
 * @param string $file 文件的本地路径。
 * @return string 用于指定文件的缓存清除值。
 */
protected function get_file_version( $file ) {
	if ( defined( 'SCRIPT_DEBUG' ) && SCRIPT_DEBUG && file_exists( $file ) ) {
		return filemtime( $file );
	}

	//就像上面所说，我们假设 WooCommerce_Example_Plugin_Assets::VERSION 解析为我们的扩展使用的某个版本号。
	return \WooCommerce_Example_Plugin_Assets::VERSION;
}
```

}

如上所述，我们需要将我们的 `IntegrationInterface` 与 WooCommerce Blocks 注册，因为我们希望在 Mini-Cart、Cart 或 Checkout 被使用时，我们的脚本被包含。因此，我们需要注册三个操作的回调函数。

```php
add_action(
    'woocommerce_blocks_mini-cart_block_registration',
    function( $integration_registry ) {
        $integration_registry->register( new WooCommerce_Example_Plugin_Integration() );
    }
);
add_action(
```

```
    'woocommerce_blocks_cart_block_registration',
    function( $integration_registry ) {
        $integration_registry->register( new WooCommerce_Example_Plugin_Integration() );
    }
);
add_action(
    'woocommerce_blocks_checkout_block_registration',
    function( $integration_registry ) {
        $integration_registry->register( new WooCommerce_Example_Plugin_Integration() );
    }
);
```

现在，当我们加载包含任何一个块的网页时，我们应该看到在 `initialize` 中注册的脚本被加载！

获取 `get_script_data` 中添加的数据

我们在 `get_script_data` 方法的 界面 中，将一些 数据 与扩展关联起来，我们需要知道如何获取这些 数据！

在 `@woocommerce/settings` 包中，有一个名为 `getSetting` 的方法，您可以导入它。 此方法 接受一个字符串 。包含在 `get_script_data` 中添加的 数据 的设置 名称，是您的 集成 的名称（即 `get_name` 返回的价值），并在其后加上 `_data`。在我们的 例子 中，它将是: `woocommerce-example-plugin_data`。

这里返回的 价值 是一个普通的 JavaScript 对象，其键由 `get_script_data` 返回的数组中的键组成，这些 值将被序列化。

版本号 #2
由 壳壳虫 创建于 2026-03-09 11:53:58 UTC
由 壳壳虫 更新于 2026-03-09 11:53:58 UTC