如何管理 WooCommerce 数据存储

目录

- 简介
- 结构
- 替换数据存储
- 创建新的数据存储
  - 定义新的产品类型
  - 用于自定义数据的存储
- 调用数据存储
- 数据存储的限制和 WP 管理员

简介

数据存储类充当 WooCommerce 的数据 CRUD 类（ `WC_Product` , `WC_Order` , `WC_Customer` 等）与数据库层之间的桥梁。 通过将数据库逻辑与数据分离， WooCommerce 变得更加灵活。 随 WooCommerce 核心提供的、基于 WordPress 的自定义文章系统和一些自定义表的的数据存储，可以被替换为不同的数据库结构、类型，甚至可以由外部 API 提供。

本指南将介绍数据存储类的结构，如何创建新的数据存储，如何替换核心数据存储，以及如何在您自己的代码中调用数据存储。

本指南中的示例将查看 `WC_Coupon` CRUD 数据类，以及 `WC_Coupon_Data_Store_CPT` ，这是一个使用 WordPress 自定义文章类型的优惠券数据存储的实现。 这是 WooCommerce 当前存储优惠券的方式。

在使用数据存储时，需要了解 `WC_Coupon` 或任何其他 CRUD 数据类包含哪些属性（ properties）。 这些属性在每个类的 `data` 数组中定义。

结构

每个 CRUD 对象的数据库存储都应该实现 `WC_Object_Data_Store_Interface` 界面。

`WC_Object_Data_Store_Interface` 包含以下方法：

- `create`
- `read`
- `update`
- `delete`
- `read_meta`
- `delete_meta`
- `add_meta`
- `update_meta`

`create`、`read`、`update` 和 `delete` 方法应该处理您属性的 CRUD 逻辑：

- `create` 应该在数据库中创建一个新的条目。 示例：创建优惠券。
- `read` 应该从数据库中查询单个条目，并根据响应设置属性。 示例：读取优惠券。
- `update` 应该对现有条目进行更改。 示例：更新或编辑优惠券。
- `delete` 应该从数据库中移除一个条目。 示例：删除优惠券。

所有数据库存储都必须实现对这些方法的处理。

除了处理您的属性之外，还可以传递其他自定义数据。 这被认为是 “元数据”。
例如，优惠券可以具有插件提供的自定义数据。

`read_meta`、`delete_meta`、`add_meta` 和 `update_meta`
方法应该被定义，以便可以从正确的来源读取和管理元数据。 在我们的 WooCommerce 核心类中，我们在
`WC_Data_Store_WP` 中定义它们，然后为所有数据库存储使用相同的代码。 它们都使用 WordPress
的元数据系统。 如果元数据应该来自不同的来源，您可以重新定义这些方法。

您的数据库存储还可以实现其他方法来替换直接查询。 例如，优惠券数据库存储具有一个公共的
`get_usage_by_user_id` 方法。
数据库存储应该始终定义和实现一个界面，用于它们期望的方法，以便其他开发者知道他们需要编写哪些方法
。换句话说，除了 `WC_Object_Data_Store_Interface` 界面之外， `WC_Coupon_Data_Store_CPT` 还实现了
`WC_Coupon_Data_Store_Interface`。

替换数据存储

让我们看看如何用 `WC_Coupon_Data_Store_Custom_Table` 类替换 `WC_Coupon_Data_Store_CPT` 类。
我们的示例将仅提供占位函数，而不是完整的解决方案。 假设我们希望将优惠券存储在名为 `wc_coupons`
的表中，该表具有以下列：

```
id, code, amount, date_created, date_modified, discount_type, description, date_expires,
usage_count,individual_use, product_ids, excluded_product_ids, usage_limit,
usage_limit_per_user, limit_usage_to_x_items, free_shipping, product_categories,
excluded_product_categories, exclude_sale_items, minimum_amount, maximum_amount,
email_restrictions, used_by
```

这些列名与属性名一一对应。

首先，我们需要创建一个新的数据存储类来包含我们的逻辑：

```
/**
 * WC 优惠券数据存储：自定义表。
 */
class WC_Coupon_Data_Store_Custom_Table extends WC_Data_Store_WP implements
WC_Coupon_Data_Store_Interface, WC_Object_Data_Store_Interface {

}
```

请注意，我们实现了主要的 `WC_Object_Data_Store_Interface` 接口以及 `WC_Coupon_Data_Store_Interface` 接口。 它们共同代表了我们需要提供逻辑的所有方法。

然后，我们将定义这些属性的 CRUD 处理：

```
/**
 * 用于在数据库中创建新优惠券的方法。
 *
 * @param WC_Coupon
 */
public function create( &$coupon ) {
	$coupon->set_date_created( current_time( 'timestamp' ) );

	/**
	 * 这是插入新优惠券的代码所在的位置。
	 * 将使用 getter 构建查询：$coupon->get_code(), $coupon->get_description() 等。
	 * 在 INSERT 操作之后，我们希望将新的 ID 传递给优惠券对象。
	 */
	$coupon->set_id( $coupon_id );

	//在创建或更新条目后，我们需要同时触发 'meta' 的保存。
	$coupon->save_meta_data();
}
```

```
// `apply_changes`方法告诉对象，当前对象反映了数据库，并且两个对象之间不存在“更改”。
$coupon->apply_changes();

//为了方便插件与您的数据存储进行交互，建议在操作完成后提供相同的钩子。
do_action( 'woocommerce_new_coupon', $coupon_id );
}

/**
 * 用于读取优惠券的方法。
 *
 * @param WC_Coupon
 */
public function read( &$coupon ) {
$coupon->set_defaults();
```

```
// `Read` 方法应该检查是否为有效的优惠券，
// 如果不是，则抛出 "Invalid coupon" 异常。
// 对于有效的优惠券，将 `$data` 设置为包含数据库结果。
// 所有属性都应使用 `set_props` 方法，并使用来自数据库的输出进行设置。这会将 CRUD 数据对象进行“填充”。
$coupon_id = $coupon->get_id();
$coupon->set_props( array(
    'code'                       => $data->code,
    'description'                => $data->description,
    // ..
) );

// 此外，我们还需要将元数据读取到对象中。
$coupon->read_meta_data();

// 此标志指示对象是否已填充。如果此标志为 false，则表示数据库未正确设置对象。
$coupon->set_object_read( true );
do_action( 'woocommerce_coupon_loaded', $coupon );
}

/**
 * 更新数据库中的优惠券。
 *
 * @param WC_Coupon
 */
public function update( &$coupon ) {
```

```
    // 更新优惠券查询，使用 getter 方法。

    $coupon->save_meta_data();
    $coupon->apply_changes();
    do_action( 'woocommerce_update_coupon', $coupon->get_id() );
}

/**
 * 从数据库中删除优惠券。
 *
 * @param WC_Coupon
 * @param array $args 传递给删除方法的参数数组。
 */
public function delete( &$coupon, $args = array() ) {
    // WordPress 和 WooCommerce 中的许多对象
    // 支持“回收站”的概念。这通常是一个标志，用于将对象
    // 移动到“回收站”。由于优惠券支持回收站，因此您的层也应该支持。
    // 如果实际发生删除，则将优惠券 ID 设置为 0。

    $args = wp_parse_args( $args, array(
        'force_delete' => false,
    ) );

    $id = $coupon->get_id();

    if ( $args['force_delete'] ) {
        // 删除查询
        $coupon->set_id( 0 );
        do_action( 'woocommerce_delete_coupon', $id );
    } else {
        // 回收站查询
        do_action( 'woocommerce_trash_coupon', $id );
    }
}
```

我们正在扩展 `WC_Data_Store_WP`，因此我们的类将继续使用 WordPress 的元数据系统。

如“结构 ”部分中所述，我们负责实现 `WC_Coupon_Data_Store_Interface` 定义的方法。每个接口描述了它接受的方法和参数，以及您的函数应该执行的操作。

一个优惠券的替换方法如下所示：

```
/**
 * 增加当前优惠券的使用次数。
 *
 * @param WC_Coupon
 * @param string $used_by 用户 ID 或账单电子邮件。
 */
public function increase_usage_count( &$coupon, $used_by = '' ) {

}

/**
 * 减少当前优惠券的使用次数。
 *
 * @param WC_Coupon
 * @param string $used_by 用户 ID 或账单电子邮件。
 */
public function decrease_usage_count( &$coupon, $used_by = '' ) {

}

/**
 * 获取特定用户 ID 的优惠券使用次数。
 *
 * @param WC_Coupon
 * @param id $user_id
 * @return int
 */
public function get_usage_by_user_id( &$coupon, $user_id ) {

}

/**
 * 返回特定 ID 的优惠券代码。
 * @param int $id
 * @return string 优惠券代码
 */
 public function get_code_by_id( $id ) {

}
```

/**

- 返回指定优惠券代码的 ID 数组。
- 可以返回多个 ID 以检查是否存在。
- @param string $code
- @return array ID 数组。 */ public function get_ids_by_code($code) {

}

一旦所有的数据存储方法都定义完成，并且逻辑编写完毕，我们需要告诉 WooCommerce 加载我们的新类，而不是内置的类。这通过使用 `woocommerce_data_stores` 过滤器来实现。一个数据存储别名数组映射到默认的 WooCommerce 类。示例：

```php
'coupon'               => 'WC_Coupon_Data_Store_CPT',
'customer'             => 'WC_Customer_Data_Store',
'customer-download'    => 'WC_Customer_Download_Data_Store',
'customer-session'     => 'WC_Customer_Data_Store_Session',
'order'                => 'WC_Order_Data_Store_CPT',
'order-refund'         => 'WC_Order_Refund_Data_Store_CPT',
'order-item'           => 'WC_Order_Item_Data_Store',
'order-item-coupon'    => 'WC_Order_Item_Coupon_Data_Store',
'order-item-fee'       => 'WC_Order_Item_Fee_Data_Store',
'order-item-product'   => 'WC_Order_Item_Product_Data_Store',
'order-item-shipping' => 'WC_Order_Item_Shipping_Data_Store',
'order-item-tax'       => 'WC_Order_Item_Tax_Data_Store',
'payment-token'        => 'WC_Payment_Token_Data_Store',
'product'              => 'WC_Product_Data_Store_CPT',
'product-grouped'      => 'WC_Product_Grouped_Data_Store_CPT',
'product-variable'     => 'WC_Product_Variable_Data_Store_CPT',
'product-variation'    => 'WC_Product_Variation_Data_Store_CPT',
'shipping-zone'        => 'WC_Shipping_Zone_Data_Store',
```

我们希望专门针对优惠券数据存储，因此我们可以这样做：

```
function myplugin_set_wc_coupon_data_store( $stores ) {
$stores['coupon'] = 'WC_Coupon_Data_Store_Custom_Table';
return $stores;
}

add_filter( 'woocommerce_data_stores', 'myplugin_set_wc_coupon_data_store' );
```

然后，WooCommerce 核心将加载我们的类，而不是 `WC_Coupon_Data_Store_CPT`。

创建新的数据存储

定义新的产品类型

你的扩展是否创建了一种新的产品类型？每一个产品类型都有一个数据存储，除了父产品的数据存储之外。
父存储负责处理共享的属性，例如名称或描述，而子存储负责处理更具体的数据。

例如，外部产品数据存储负责处理“按钮文本”和“外部 URL”。变量数据存储负责父产品与其变体之间的关系。

请查看此教程，以获取有关此过程的更多信息。

用于自定义数据的存储

如果你的扩展引入了新的数据库表、新的自定义文章类型，或者某种与产品、订单等无关的新数据形式，那么
你应该实现你自己的数据存储。

你的数据存储仍然应该实现 `WC_Object_Data_Store_Interface` 接口，并提供标准的 CRUD 函数。
你的数据存储应该是与你的数据交互的主要入口点，因此任何其他查询或操作也应该有相应的方法。

配送区域数据存储是一个很好的例子，它展示了如何使用自定义表创建一个“简单”的数据存储。
优惠券的代码是一个很好的例子，展示了如何使用自定义文章类型的数据存储。

要注册你的数据存储，你只需要将其添加到 `woocommerce_data_stores` 过滤器中：

```
function myplugin_set_my_custom_data_store( $stores ) {
	$stores['mycustomdata'] = 'WC_My_Custom_Data_Store';
	return $stores;
}

add_filter( 'woocommerce_data_stores', 'myplugin_set_my_custom_data_store' );
```

然后，你可以像加载其他 WooCommerce 数据存储一样加载你的数据存储。

调用数据存储

调用数据存储就像使用静态的 `WC_Data_Store::load()` 方法一样简单：

```
// 加载配送区域数据存储。
$data_store = WC_Data_Store::load( 'shipping-zone' );
// 获取配送区域 ID 为 4 的配送方法数量。
$num_of_methods = $data_store->get_method_count( 4 );
```

你也可以链式调用方法：

```
// 获取配送区域 ID 为 4 的配送方法数量。
$num_of_methods = WC_Data_Store::load( 'shipping-zone' )->get_method_count( 4 );
```

`::load()` 方法适用于注册到 `woocommerce_data_stores` 的任何数据存储，因此你可以加载你的自定义数据存储：

```
$data_store = WC_Data_Store::load( 'mycustomdata' );
```

数据存储的限制和 WP 管理员

目前，一些 WooCommerce 界面仍然依赖于 WordPress 来列出对象。例如，优惠券和产品。

如果您通过数据存储进行数据替换，那么现有用户界面的一些部分可能会出现故障。例如，当使用 `type` 过滤器时，优惠券列表可能会出现问题。此过滤器使用元数据，然后将其传递给 WordPress，WordPress 使用 `WP_Query` 类执行查询。`WP_Query` 类无法处理位于常规元数据表之外的数据（参考 #19937）。为了解决这个问题，需要逐步淘汰 `WP_Query` 的使用，并用自定义的查询类和函数进行替换。

版本号 #2
由 壳壳虫 创建于 2026-03-09 06:29:41 UTC
由 壳壳虫 更新于 2026-03-09 06:29:41 UTC