

# POD 技术方案

## 开发流程概述

1. 注册（创建开发者账号、注册 Partner 账户）
2. 开发APP&认证（获取admin API访问权限）
  - a. 创建应用：
    - i. 创建 Public App，配置 OAuth 2.0 认证。
    - ii. 获取商店 shop 和 client secret 来完成与API交互。
  - b. 开发 商品定制 等功能。
    - i. 调用 Admin API 进行数据交互。
3. 开发 Theme Extension：
  - a. 安装 Shoplazza CLI：
    - i. 使用 `npm install -g shoplazza-cli` 安装 CLI。
  - b. 创建 Theme Extension 项目：
    - i. 使用 `shoplazza te create` 创建项目。
    - ii. 选择 Extension 类型（Basic 或 Embed）。
  - c. 编写 Liquid 代码：
    - i. 在 `blocks/` 目录下编写 Liquid 文件，定义扩展功能。
    - ii. 在 `snippets/` 目录下编写可复用的 Liquid 片段。
  - d. Theme Extension测试：
    - i. 使用 `shoplazza te serve` 启动本地开发环境。
    - ii. 在主题编辑器中预览 Extension 效果。
4. 联调测试
  - a. Theme Extension 和 APP 联调测试
5. 发布 Theme Extension
6. 提交审核：
  - a. 提交 APP 审核：
    - i. 在 Shoplazza 开发者后台提交 APP 审核。

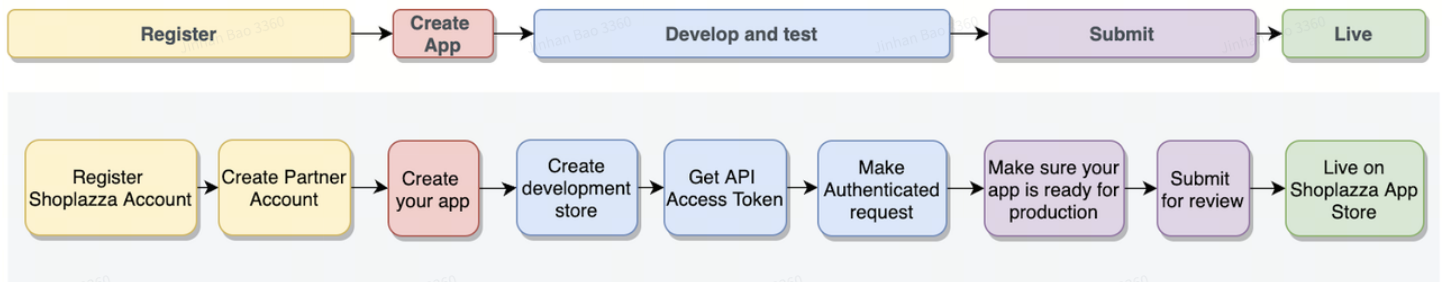
ii. 提供 APP 的功能描述、截图、使用说明等。

**b. 审核通过：**

i. Shoplazza 审核团队对 APP 和 Theme Extension 进行审核。

ii. 审核通过后，开发者会收到通知。

## 7. 发布到 APP 到 App Store



### step1：入驻

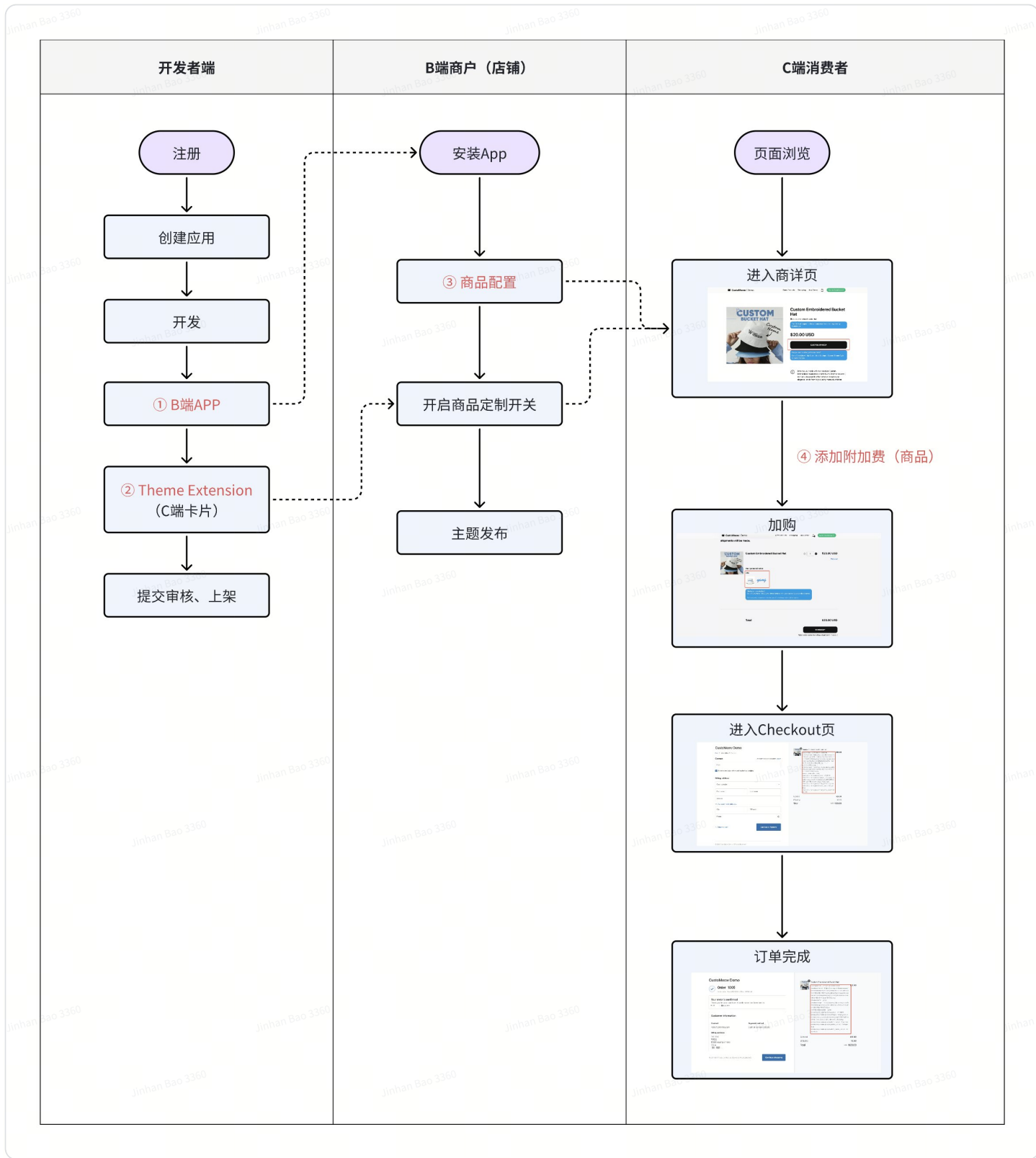
<https://www.shoplazza.dev/reference/1create-a-developer-platform-login-account-and-register-as-a-partner>

### step2：创建应用

<https://www.shoplazza.dev/reference/2create-a-public-app>

### step3：开发

### 整体流程



## B端APP相关

### Admin API 相关: Overview

1. 配置Admin API 的access token: 在调用 Admin API 之前需要完成 OAuth 2.0 Authentication 认证, 获取访问令牌

2. 在请求 API 之前，确保已开通相应的权限范围（access scope），例如：

- write\_product（写入商品数据）
- read\_product（读取商品数据）
- read\_order（读取订单数据）

参考文章：[如何开通相关权限](#) [access\\_scopeA 合集](#)

这些 API 会在开发场景中使用

- Product API: [Product API 参考文档](#)（用于获取、创建、更新、删除商品信息）
- Order API: [订单API文档](#)（用于查询、管理订单信息）
- Metafield API : [META FIELD](#)（用于存储和管理商品的自定义字段。）

类目	分类目	场景	API
Product	Product	Create product	<a href="#">VIEW</a>
		Update product	<a href="#">VIEW</a>
		Cancel product	<a href="#">VIEW</a>
		product Details	<a href="#">VIEW</a>
		product List	<a href="#">VIEW</a>
	variant	Create variant	<a href="#">VIEW</a>
		Get variant Detail	<a href="#">VIEW</a>
		Get variant List	<a href="#">VIEW</a>
Order	Order	Order List	<a href="#">查看</a>
		Order Detail	<a href="#">查看</a>
metafields	metafields	Create Matefield	<a href="#">查看</a>
		Get Metafield Details	<a href="#">查看</a>
		Get Metafield List	<a href="#">查看</a>
		Update Matefield	<a href="#">查看</a>

## Webhooks :

Shoplazza 支持以 Webhook 的形式监听特定事件（如订单创建、商品变更），目前支持的 webhook 事件：[webhook 事件列表](#)

App 可以通过 [OpenAPI](#) 注册 webhook 事件。接受 webhook 时，需要验证请求的安全性，具体参考：[验证webhook](#)

在 POD 插件需要用到的 Webhooks :

- [商品 product Webhook文档](#)（监听商品相关的变更事件）
- [购物车cart webhook 文档](#)（监听购物车相关的变更事件）

## 后端 OAuth 2.0 认证流程

### 1. 安装依赖

Code block

```
1 npm install express crypto axios
```

### 2. 配置文件 /config/index.js

用于存放 APP\_NAME, CLIENT\_ID, CLIENT\_SECRET, REDIRECT\_URI

Code block

```
1 const CLIENT_ID = "<YOUR_CLIENT_ID>";
2 const CLIENT_SECRET = "<YOUR_CLIENT_SECRET>";
3 const BASE_URL = "<BASE_URL>";
4 const REDIRECT_URI = `${BASE_URL}/auth/shoplazza/callback`;
5 let access_token = {};
```

### 3. HMAC 校验中间件 /middleware/hmacValidator.js

Code block

```
1 import crypto from "crypto";
2
3 export const hmacValidator = async (ctx, next) => {
4   const { hmac, ...queryParams } = ctx.query;
5
6   if (!hmac) {
7     ctx.status = 400;
8     ctx.body = { error: "Missing HMAC parameter" };
9     return;
10  }
```

```

10   }
11
12   // 计算 HMAC 校验
13   const message = Object.keys(queryParams)
14     .sort()
15     .map((key) => `${key}=${queryParams[key]}`)
16     .join("&");
17
18   const generatedHmac = crypto
19     .createHmac("sha256", process.env.CLIENT_SECRET)
20     .update(message)
21     .digest("hex");
22
23   if (crypto.timingSafeEqual(Buffer.from(generatedHmac), Buffer.from(hmac))) {
24     await next();
25   } else {
26     ctx.status = 403;
27     ctx.body = { error: "HMAC validation failed" };
28   }
29 };

```

#### 4. 认证路由 /routes/auth.js

Code block

```

1   import Router from "koa-router";
2   import crypto from "crypto";
3   import axios from "axios";
4   import { APP_NAME, CLIENT_ID, CLIENT_SECRET, REDIRECT_URI } from
    "../config/index.js";
5   import { hmacValidator } from "../middleware/hmacValidator.js";
6
7   const router = new Router({ prefix: "/api" });
8
9
10  router.get(`/auth/install`, async (ctx) => {
11    const shop = ctx.query.shop;
12    if (!shop) {
13      ctx.status = 400;
14      ctx.body = { error: "Missing shop parameter" };
15      return;
16    }
17
18    const scopes = "read_customer,read_product,write_product";
19    const state = crypto.randomBytes(16).toString("hex");
20    const redirectUri = `https://${ctx.host}${REDIRECT_URI}`;

```

```

21
22   const authUrl = `https://${shop}/admin/oauth/authorize?
client_id=${CLIENT_ID}&scope=${scopes}&redirect_uri=${redirectUri}&response_type=code&state=${state}`;
23
24   ctx.redirect(authUrl);
25 });
26
27 // ** Step 2: 处理 OAuth 回调**
28 router.get(`/auth/callback`, hmacValidator, async (ctx) => {
29   const { code, shop } = ctx.query;
30
31   if (!shop || !code) {
32     ctx.status = 400;
33     ctx.body = { error: "Missing required parameters" };
34     return;
35   }
36
37   const redirectUri = `https://${ctx.host}${REDIRECT_URI}`;
38
39   try {
40     const response = await axios.post(`https://${shop}/admin/oauth/token`, {
41       client_id: CLIENT_ID,
42       client_secret: CLIENT_SECRET,
43       code,
44       grant_type: "authorization_code",
45       redirect_uri: redirectUri,
46     });
47
48     console.log("获取 access_token 成功:", response.data);
49     ctx.body = response.data; // 返回 token 及 store 信息
50   } catch (error) {
51     console.error("获取 access_token 失败:", error.message);
52     ctx.status = 500;
53     ctx.body = { error: "Failed to obtain access token" };
54   }
55 });
56
57 export default router;

```

## 5. 启动服务器挂载oauth，认证路由

### Code block

```

1  const PORT = process.env.PORT || 3000;
2  app.listen(PORT, () => {

```

```
3 console.log(`Server running on http://localhost:${PORT}`);
4 });
```

## Pod 需要完成的关键功能：

1. 商品列表展示
2. 创建定制商品（使用tag 字段，标记定制商品，例如 存储 ‘customization’ 字段到tags）
3. 创建Metafield Definition 和 Metafield（将定制商品 ID 关联到原商品）
4. 获取 Metafield 数据（展示关联的定制商品）

代码示例：

### 1. 获取商品列表（过滤 tag 为 customization 商品）

Code block

```
1 const router = new Router({ prefix: "/api/products" });
2
3 /**
4  * **获取商品列表**
5  * - 过滤掉 "customization" 标签的商品
6  */
7 router.get("/", async (ctx) => {
8   try {
9     const response = await
10     axios.get(`https://${SHOP}/openapi/${API_VERSION}/products`, {
11       headers: { "access-token": ACCESS_TOKEN },
12     });
13     const products = response.data.products.filter(p => !(p.tags ||
14     []).includes("customization"));
15     ctx.body = { products };
16   } catch (error) {
17     console.error("获取商品失败:", error.message);
18     ctx.status = 500;
19     ctx.body = { error: "获取商品失败" };
20   }
21 });
```

### 2. 创建定制商品（使用tag 字段，标记定制商品，例如 存储 ‘customization’ 字段到tags）

Code block

```
1 const createCustomization = await
```



```

1 axios.post(`https://${SHOP}/openapi/${API_VERSION}/products`, {
2     product: {
3         has_only_default_variant: true,
4         title,
5         brief,
6         description,
7         published: false,
8         requires_shipping: false,
9         taxable: false,
10        inventory_policy: "continue",
11        tags: "customization",
12        images,
13        options,
14        variants
15    }
16 }, {
17     headers: { "access-token": ACCESS_TOKEN }
18 });

```

### 3. 创建Metafield Definition 和 Metafield（将定制商品 ID 关联到原商品）

Code block

```

1 await
2 axios.post(`https://${SHOP}/openapi/${API_VERSION}/product/${productId}/metafields`, {
3     definition_id: definition_id,
4     namespace: "customization",
5     key: "design",
6     value: customizationId,
7     type: "string"
8 }, {
9     headers: { "access-token": ACCESS_TOKEN }
10 });

```

### 4. 获取 Metafield 数据（展示关联的定制商品）

Code block

```

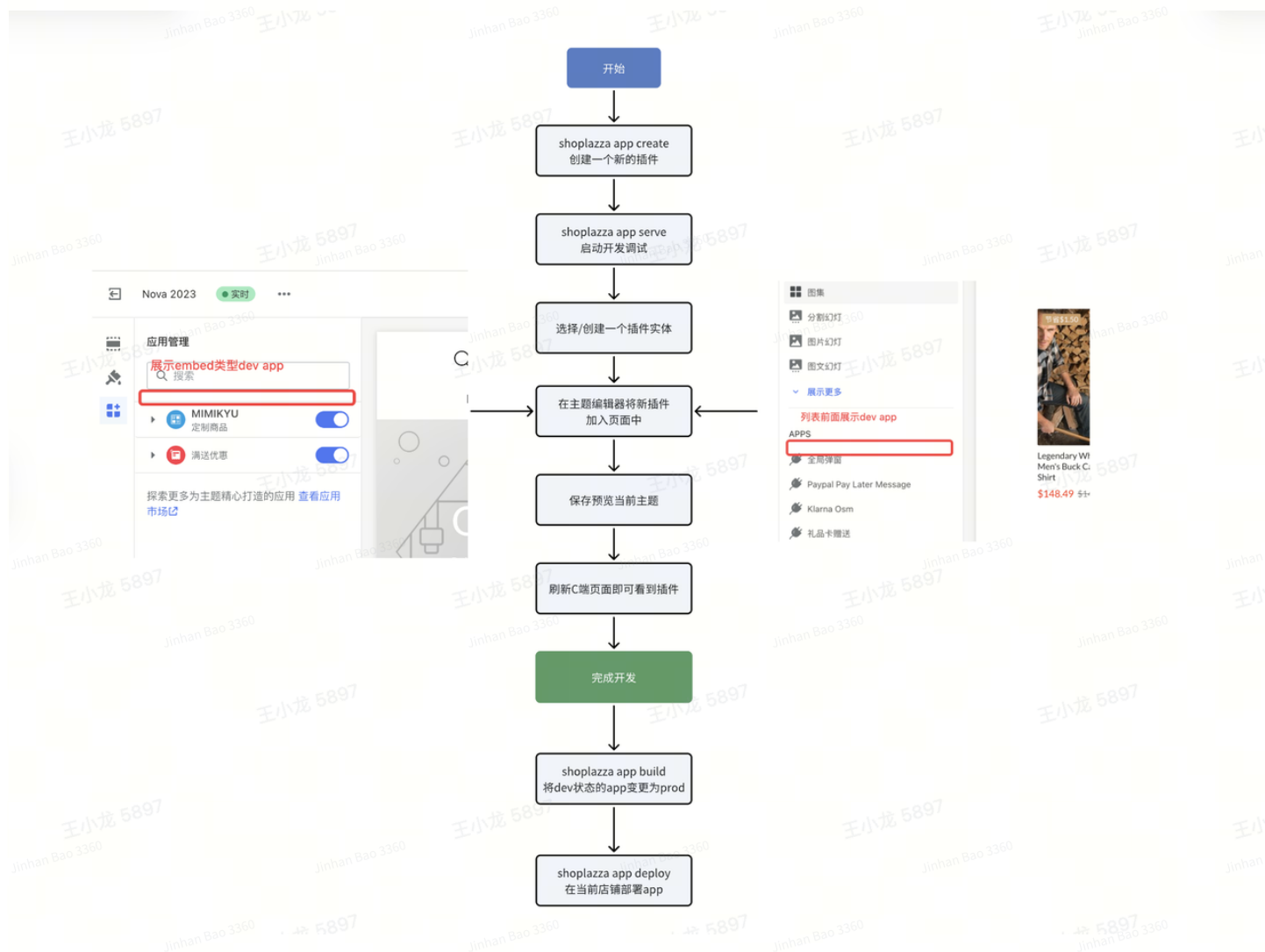
1 await
2 axios.get(`https://${SHOP}/openapi/${API_VERSION}/product/${productId}/metafields`, {
3     headers: { "access-token": ACCESS_TOKEN },
4 });

```

# Theme Extension开发相关

Theme Extension是店匠提供的针对主题的扩展方式，开发者可以通过开发Theme Extension来完成对主题页面的扩展和定制。

## 开发流程



## Cli 相关资料:

- [shoplazza-cli](#)
- Cli command 合集: [shoplazza-cli](#) -> 目录: Theme commands

## 快速开始

## 安装

Code block

```
1 npm install -g shoplazza-cli
```

## 登陆开发者店铺

Code block

```
1 shoplazza login --store {shopdomain}.myshoplaza.com
```

## 创建 Extension

Code block

```
1 shoplazza te create
```

创建新Extension会提示选择的主题插件类型，存在两种类型：Basic Extension 和 Embed Extension。

- Basic Extension支持在主题编辑器中调整插入的位置，
- Embed Extension的插入位置则是硬编码在Extension中，插入位置固定（推荐 pod 开发者 使用）

执行命令成功后会[创建一个新项目](#)

## 启动开发

备注：开始开发之前请要先给您的商店建立主题，且要应用主题

Code block

```
1 cd your-project
2 shoplazza te serve
```

因为插件展示在主题页面上，所以启动时需要选择一个主题进行预览，

```
┌─┴─┐ ┌─┴─┐ ┌─┴─┐ ┌─┴─┐ ┌─┴─┐ ┌─┴─┐ ┌─┴─┐ shoplazza te serve
[WARNING]the version of current Node.js is v22.14.0, advise select the support version: v20.9.0,v20.18.0
✓ Successfully synchronized local files to remote location!
✓ Theme list loaded.
? Please select a theme to preview: Copy of Nova 2023
```

在选择之后会展示预览链接。

```
? Please select a theme to preview: Nova 2023
|

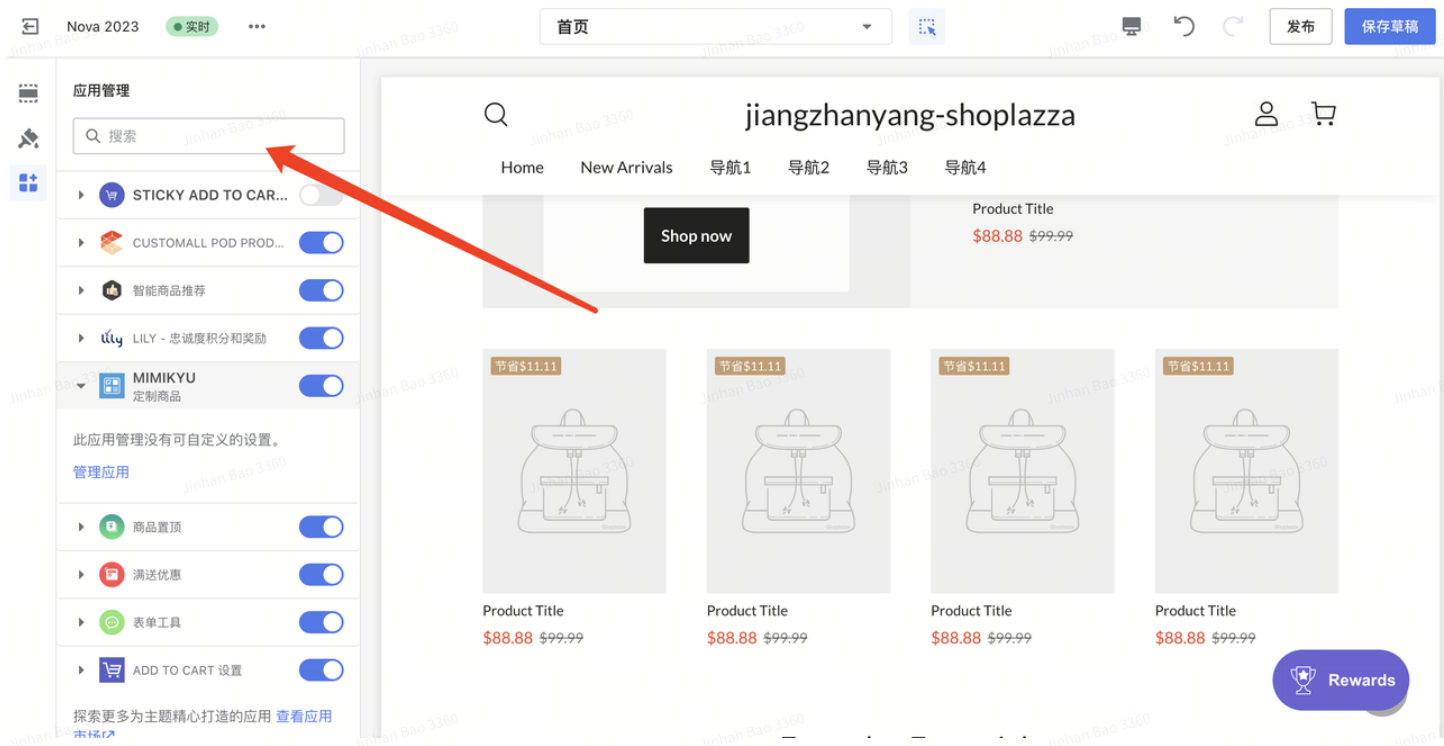
=====PREVIEW URLS=====

🔗 Admin Preview URL:
https://locke.dev.myshoplaza.com/admin/card?theme_id=6928713b-aeca-43dc-8382-5430f80fc1d8&

🔗 Storefront Preview URL:
https://locke.dev.myshoplaza.com?preview_theme_id=6928713b-aeca-43dc-8382-5430f80fc1d8&ext

=====WATCHER RUNNING=====
```

- 第一个链接是跳转主题编辑器，可以在编辑器上开启开关把当前插件加入到主题页面中，加入之后需要保存之后才能在C端进行预览。

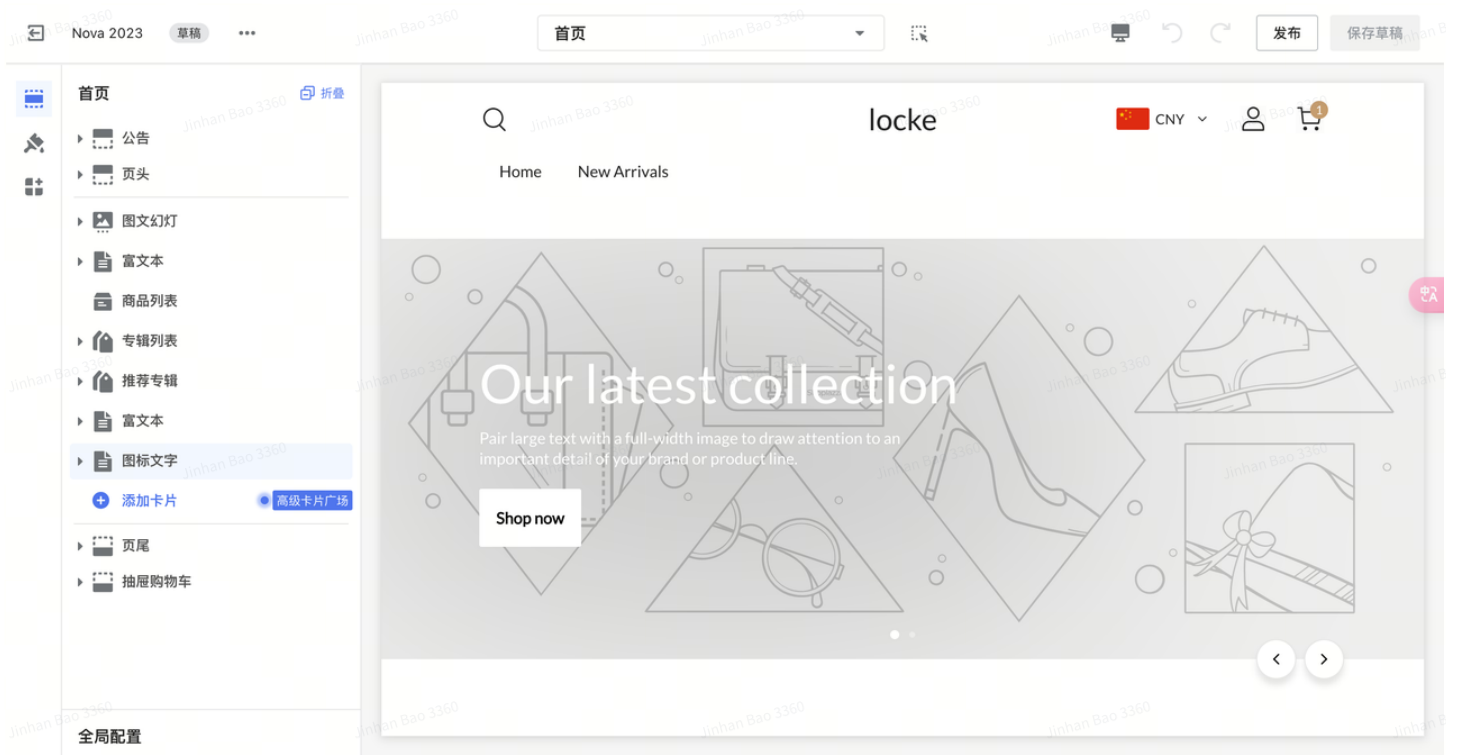


- 第二个链接是跳转C端页面进行预览，在编辑器将卡片加入到页面之后，每一次在本地更新卡片代码，刷新C端页面就能看到最新效果。

## Theme Extension 主题编辑浏览

### 预览流程

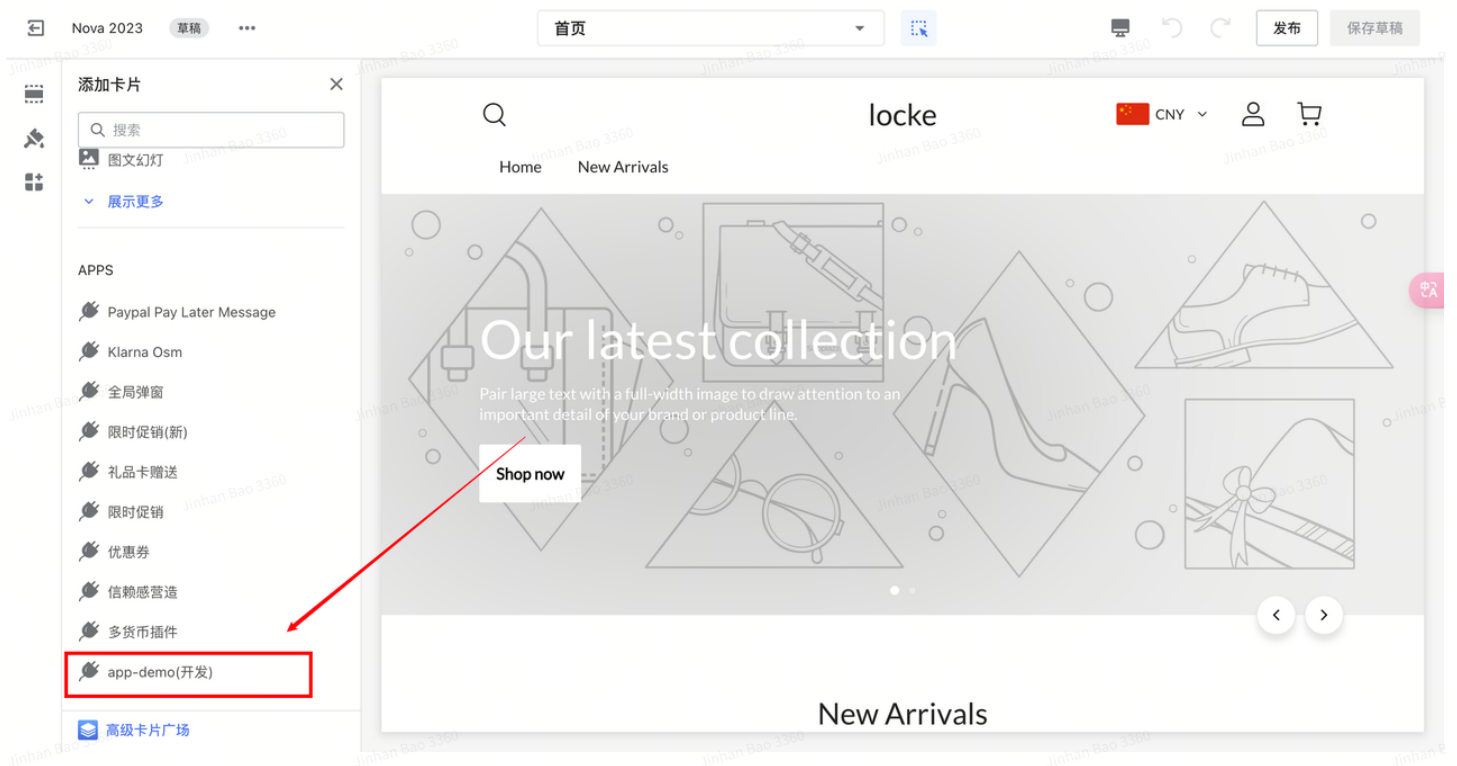
1. 浏览器访问店铺后台的主题编辑器预览链接



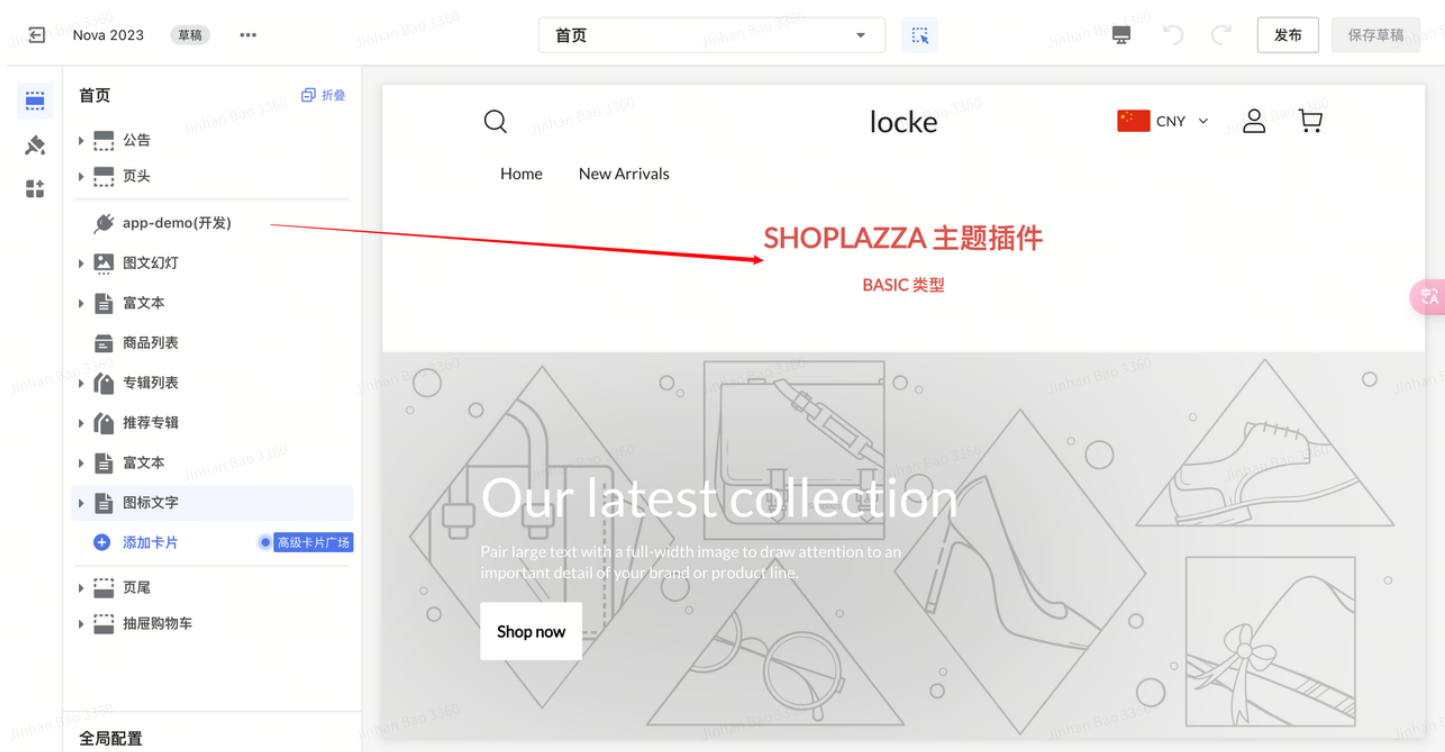
## 2. 在店铺后台的主题编辑器添加主题插件

### Basic App:

主题编辑器 -> 添加卡片 -> APPS 找到开发模式启动的主题插件

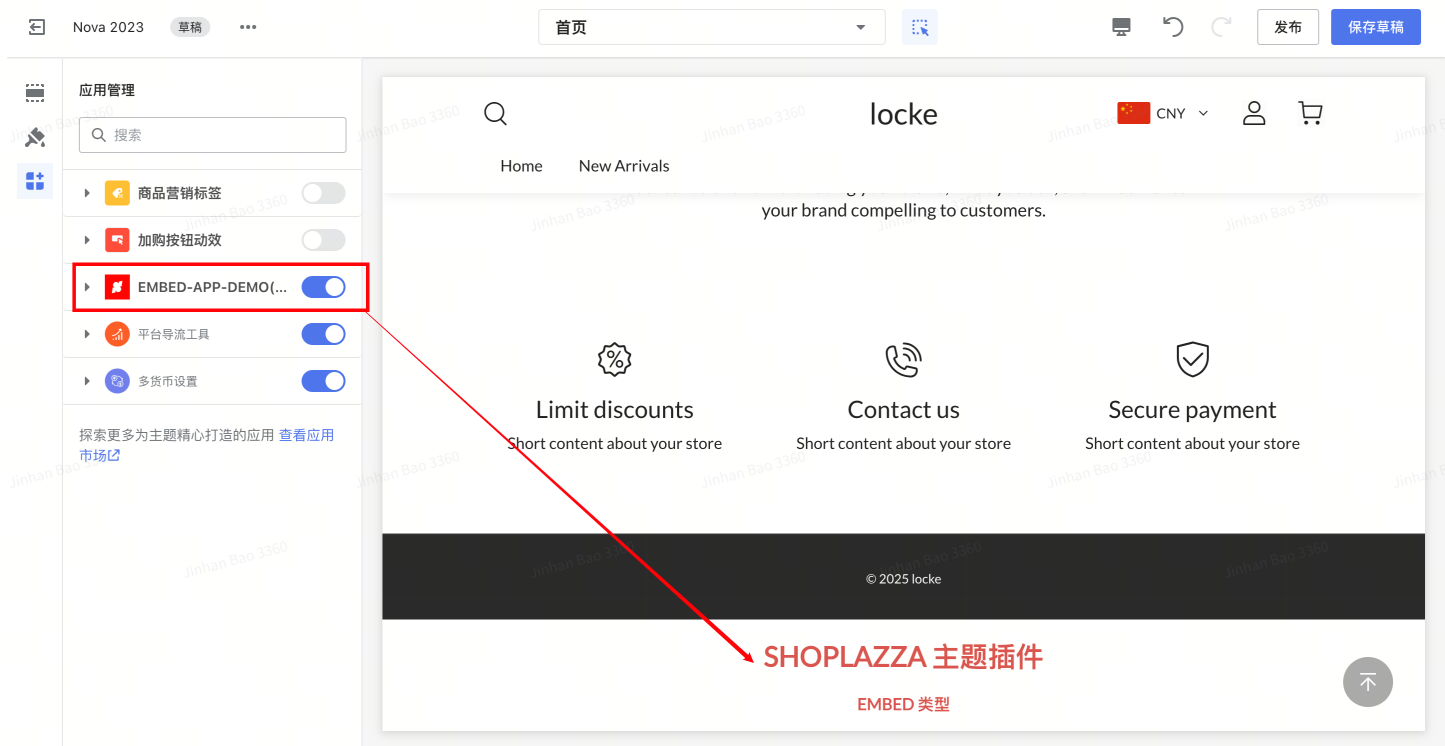


点击主题插件，即可在右侧预览页面看到插件渲染内容（在主题插件添加到主题中后，建议点击保存草稿，避免改动代码后刷新浏览器需要重复添加插件）

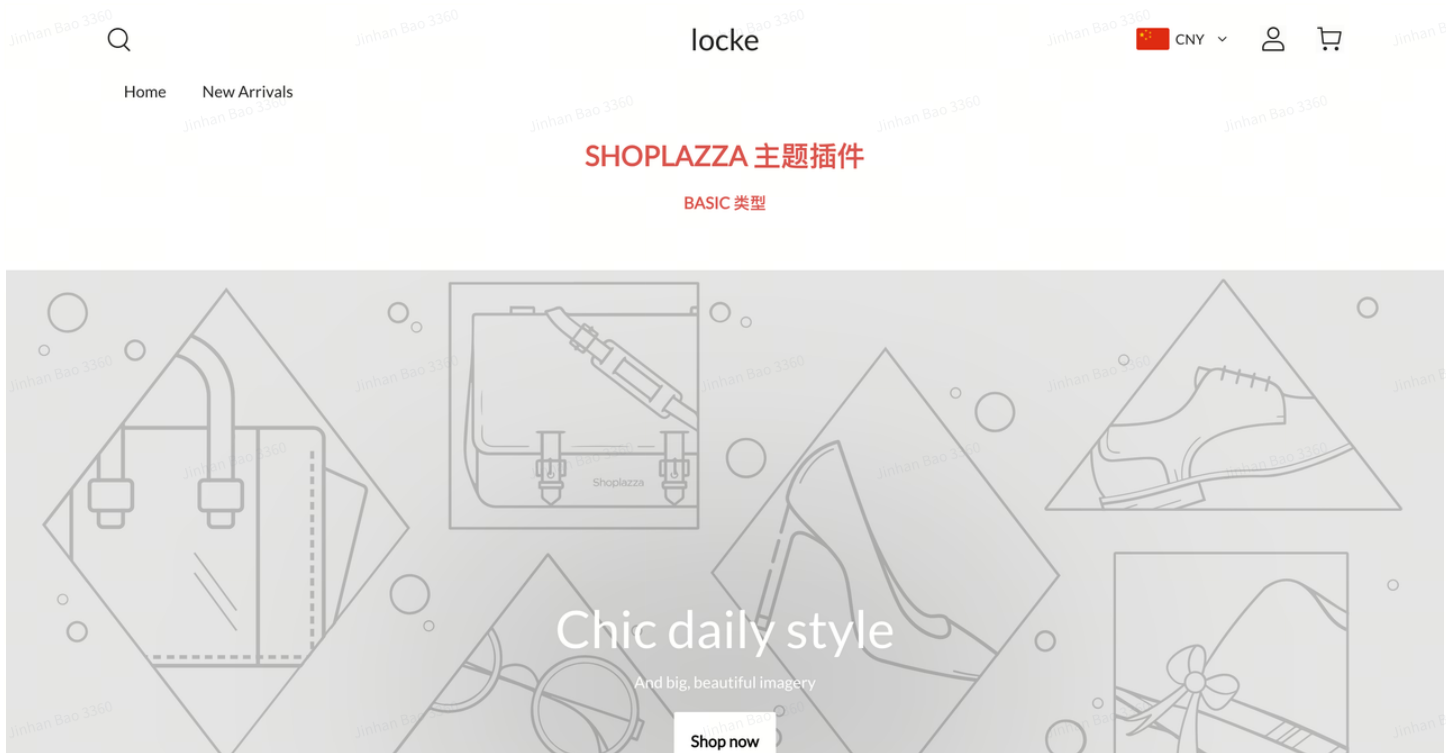


## Embed App:

应用管理，启用插件



3. 店铺后台的主题编辑器配置完成，**保存草稿**，即可访问店铺前台预览链接进行预览



## 构建正式版本

Code block

```
1 shoplazza te build
```

基于当前项目代码生成可供部署的正式版本，可以生成多个，部署的时候会选择其中一个。

## 部署Extension

当要将一个extension在当前店铺上线，可以通过部署命令。

Code block

```
1 shoplazza te deploy
```

部署成功后，该卡片将在当前店铺完成上线。

## Extension绑定APP

APP是shoplazza的主要扩展方式，Theme Extension是APP的一部分，一个APP可以绑定多个Theme Extension。当需要将当前Theme Extension绑定到指定的APP，可以执行以下命令：

Code block

```
1 shoplazza te connect
```

命令执行会提示输入APP的 `client_id` 和 `client_secret` ，这些可以从[shoplazza partner后台](#)获得。

#### Client credentials

你的App的公共标识符，这是OAuth流程以及与Shoplazza API互动所需要的。

Client ID

oV7LWEX24lldv7128Mj-X1Cr4lv7Yjmwzu3c1KC2EsQ

Client secret

.....

## 发布Extension

在绑定app之后，可以通过release命令将Extension发布出去，发布会选择当前店铺的一个正式版本。

Code block

```
1 shoplazza te release
```

发布的版本是基于当前店铺正式版本中一个，发布出去的版本可以绑定到app，以实现安装app就能安装插件。

## 技术规范

### 项目目录规范

Code block

```
1  项目名/  
2  |— theme-app/  
3  |   |— assets/  
4  |   |   |— logo.png  
5  |   |— blocks/  
6  |   |   |— subscription.liquid  
7  |   |— locales/  
8  |   |   |— en-US.json  
9  |   |   |— zh-CN.json  
10 |   |   |— zh-TW.json  
11 |   |   |— ...  
12 |   |— snippets/  
13 |   |— subscription_icon.liquid
```



```

14 | | subscription_script.liquid
15 | | ...
16 | | README.md
17

```

目录/文件	说明
<b>theme-app/</b>	存放应用的主要代码。
theme-app/assets	存放静态资源的文件，包括js、css、图片等。
theme-app/blocks	用来存放要插入主题的内容的liquid文件，其中的每一个liquid文件都是独立的block； <a href="#">App blocks for themes</a>
theme-app/snippets	用来存放封装好的 Liquid 片段 和 icon文件。 可在多个 block 中通过 {% include 'snippet文件名' %} 使用。
theme-app/locales	存放多语言文案的文件，包含15个地区(包括中国台湾)、14个国家的不同文案。

## 实现规范

Liquid	<a href="https://www.shoplazza.dev/docs/liquid-overview">https://www.shoplazza.dev/docs/liquid-overview</a>
Liquid Metafields	<a href="https://www.shoplazza.dev/docs/overview-22">https://www.shoplazza.dev/docs/overview-22</a>
Ajax API	<a href="https://www.shoplazza.dev/docs/overview-13">https://www.shoplazza.dev/docs/overview-13</a>
JS全局对象	<ul style="list-style-type: none"> <li>建议直接使用Liquid；</li> <li>JS中使用window.C_SETTINGS</li> </ul>
链接标准写法（跳转、请求）	<p><b>Locale-aware URLs</b></p> <p>Stores can have dynamic URLs generated for them when they sell internationally or in multiple languages. When using the Ajax API, it's important to use dynamic, locale-aware URLs so that you can give visitors a consistent experience for the language and country that they've chosen.</p> <p>The global value <code>window.SHOPLAZZA.routes.root</code> is available to use as a base when building locale-aware URLs in JavaScript. The global value will always end in an empty string, so you can safely use simple string concatenation to build the full URLs.</p> <pre> Code block 1 // Ajax API 2 fetch(window.SHOPLAZZA.routes.root + '/api/cart') 3   .then(response =&gt; response.json()) 4   .then(data =&gt; { /* do something */}); </pre>

	<div>Code block</div> <pre>1 &lt;!-- Liquid --&gt; 2 &lt;a href="{ ' /page/custom-page'   add_root_url   }"&gt;CUstom Page&lt;/a&gt;</pre>
Deeplink - 深度链接	<a href="#">📄 Block Deeplink 自动安装引导</a> <span>（内部文档）</span>

## 添加附加费（商品）

附加费是以商品形式呈现。在用户执行加购操作时，监听到用户的行为，对表单提交事件进行拦截处理。

拦截后，系统会弹出定制配置弹框，此弹框供用户依据实际情况完成相关配置。

待用户完成配置后，点击弹框中的“添加”按钮，系统将按照配置内容对商品属性进行更改，最终完成商品加购流程

## 监听并拦截加购事件

拦截加购事件是以注册**JS Hook**解决的方式，点击加购按钮后触发弹框。（**JS Hook**允许开发者在加购、下单等关键流程中拦截并修改数据。）

示例：

Code block

```
1 window.addEventListener("load", () => {
2   if (typeof SPZServices !== 'undefined') {
3     SPZServices.registerFor('product-form').modify((productData) => {
4       return new Promise((resolve, reject) => {
5         console.log(' 收到原始加购数据:', productData);
6
7         // 如果 productData 是数组，取第一个元素
8         if (Array.isArray(productData)) {
9           productData = productData[0];
10        }
11
12        // 显示弹窗
13        const popup = document.getElementById('cart-popup');
14        popup.style.display = 'block';
15
16        // 获取商品 ID
```

```

17     const productId = productData.product_id;
18     console.log('🔍 正在获取商品信息, ID:', productId);
19
20     // 后续逻辑...
21     genProperties().then((customProperties) => {
22         resolve({
23             properties: customProperties
24         });
25     });
26 });
27 }
28 });

```

## 调用函数：

```
SPZServices.registerFor({transitionName}).modify( {callback})
```

注册一个 Hook，监听加购事件。当前函数可以提供对加购、下单等流程的数据校验、数据修改能力；

## 参数

- {transitionName}：在当前场景固定为'product-form'，标识向“商品表单”中注册逻辑；
- `ProductFormData`：加购事件触发时传递的商品数据（加购请求API 的请求参数 - [add to cart API request](#)）
  - productData 在“buy now”事件中为一个商品数据的数组，需要取第一个元素
  - productData 在"add to cart"事件中为一个商品的object。
- `Promise`：返回一个 Promise，用于控制加购流程的继续或中断

json

```

1  {
2      "product_id": "b1ab6324-ae4f-4cca-907e-18af16839447",
3      "variant_id": "4a3d825c-2fac-4f72-a68b-b8604b0935c4",
4      "quantity": 1
5  }

```

- `Properties`：返回的properties 该值 将合并到商品表单数据中，数据格式：{ properties: { [key:string]: any } }

Code block

```

1  "properties": {
2      "custom_note": "good",

```

```
3     "_hide_note": "89" // 以下划线开头的properties将会在页面隐藏，不会在C端用户侧展
    示
4 }
```

## 自定义购物车商品属性 - Properties

建议通过购物车 商品的 `Properties` 来实现自定义购物车商品属性。

**进行加购 → 商品的 properties 进入购物车 → 当订单创建后，购物车中的 properties 会传递到订单**

*Tips: 如果你不想在C端让消费者看到对应的属性，可以将属性名以 “\_” 开头。*

方式1:

参考: [spz-product-form 商品表单 - Lessjs](#)

在插件卡片中置入表单元素，卡片插入到商品详情主卡片之后，表单将会自动携带数据提交:

Code block

```
1  <!--以input为例，将原生表单元素放入 form表单重，将会自动携带至购物车商品或下单商品-->
2  <!--其中 name="properties[checkbox]" 定义了自定义属性的名称，使用时请替换-->
3  <input id="checkbox" required class="required" type="checkbox"
    name="properties[checkbox]" value="Yes" />
```

方式2:

参考: [Cart API reference](#)，将商品加入购物车，并附加定制信息：通过在加购请求体中增加 `properties` 属性值

API:

增加单个variant 到购物车: `POST /{locale}/api/cart`

批量增加variant 到购物车: `POST /{locale}/api/cart/batch`

参数示例:

Code block

```
1  {
2    "line_items": [
3      {
4        "variant_id": "199c4d2b-7dbc-43c1-baec-7b5bba113078",
5        "quantity": 1,
6        "properties": {
7          "custom_note": "good",
```

```

8      "_hide_note": "89"  // 以下划线开头的properties将会在页面隐藏，不会
    在C端用户侧展示
9      }
10   }
11 ],
12   "refer_info": {
13     "source": "cart"
14   }
15 }

```

## 获取附加费商品

附加费 商品的商品id是从**metafield**中获取的，此信息是在商品进行配置时被存储在metafield里的。对于开发者而言，需要调用**Product API**来获取商品的详细信息，进而顺利完成后续的加购操作。

### API:

获取商品详细信息 API: [Product API](#)

创建metafield的接口: [create metafield](#) (在resource 和 resource ID中绑定好对应的商品 ID)

### Metafield相关文档

- Metafields API: [Metafield API 参考文档](#) [metafields使用指南](#)
- C端获取Metafields: [metafield C端获取文档](#)

### 代码示例:

- **获取主商品信息:** 通过 `/api/products/:product_id` 接口获取主商品的详细信息。
- **获取 Metafield:** 通过 GET `/api/front/metafields/product/list?owner_ids={owner_id}&namespace={namespace}&key={key}&page={page}&limit={limit}` 接口获取元字段
- **获取附加商品信息:** 通过 `/api/products/:meta_product_id` 接口获取附加商品的详细信息。
- **更新 UI:** 将主商品和附加商品的信息动态更新到弹窗中。

#### Code block

```

1  // 获取商品信息并更新弹窗内容
2  function fetchProductDetails(productId) {
3    fetchProductInfo(productId)
4      .then(product => {
5        updateMainProductInfo(product); // 更新主商品信息
6        return fetchMetafield(productId); // 获取附加商品的 Metafield
7      })

```

```

8      .then(metafield => {
9          if (metafield) {
10             const metaProductId = extractMetaProductId(metafield); // 提取附加商品 ID
11             return fetchProductInfo(metaProductId); // 获取附加商品信息
12         }
13         throw new Error('未找到附加商品信息');
14     })
15     .then(metaProduct => {
16         updateMetaProductInfo(metaProduct); // 更新附加商品信息
17         updateTotalPrice(); // 更新总价
18     })
19     .catch(error => {
20         console.error('数据获取失败:', error);
21         handleError(); // 处理错误
22     });
23 }
24
25 // 获取商品信息
26 function fetchProductInfo(productId) {
27     return fetch(`${window.SHOPLAZZA.routes.root}/api/products/${productId}`)
28         .then(response => response.json())
29         .then(data => data.data.product);
30 }
31
32 // 获取 Metafield
33 GET /api/front/metafields/product/list?owner_ids={owner_id}&namespace=
{namespace}&key={key}&page={page}&limit={limit}
34
35 // 提取附加商品 ID
36 function extractMetaProductId(metafield) {
37     return metafield.value.trim();
38 }
39
40 // 更新主商品信息
41 function updateMainProductInfo(product) {
42     document.getElementById('main-product-img').src = product.image?.src || '';
43     document.getElementById('main-product-title').textContent = product.title ||
'未知商品';
44     document.getElementById('main-product-price').textContent = `¥
${product.price_min || '0.00'}`;
45 }
46
47 // 更新附加商品信息
48 function updateMetaProductInfo(metaProduct) {
49     document.getElementById('meta-product-img').src = metaProduct.image?.src ||
'';

```

```

50     document.getElementById('meta-product-title').textContent = metaProduct.title
    || '未知商品';
51     document.getElementById('meta-product-price').textContent = `¥
    ${metaProduct.price_min || '0.00'}`;
52 }
53
54 // 更新总价
55 function updateTotalPrice() {
56     const mainPrice = parseFloat(document.getElementById('main-product-
    price').textContent.replace('¥', '')) || 0;
57     const metaPrice = parseFloat(document.getElementById('meta-product-
    price').textContent.replace('¥', '')) || 0;
58     const total = mainPrice + metaPrice;
59     document.getElementById('total-price').textContent = `¥${total.toFixed(2)}`;
60 }
61
62 // 处理错误
63 function handleError() {
64     document.getElementById('meta-product-title').textContent = '未能获取关联商品';
65     document.getElementById('meta-product-price').textContent = '¥0.00';
66     updateTotalPrice();
67 }

```

## 加购

- 当用户完成定制选项的选择后，页面上会出现“添加”按钮。
- 监听到用户点击此按钮时，系统会调用加购接口，从而将商品成功加入到购物车中。

### API:

增加单个variant到购物车: `POST /{locale}/api/cart`

批量增加variant到购物车: `POST /{locale}/api/cart/batch`

### 代码示例:

- **准备加购数据:** 构造一个包含主商品和附加商品的加购请求体。
- **调用批量加购 API:** 使用 `POST /api/cart/batch` 接口将商品加入购物车。
- **处理加购结果:** 根据加购结果提示用户，并决定是否继续加购流程。

#### Code block

```

1     document.getElementById('confirm-add').addEventListener('click', () => {
2         const popup = document.getElementById('cart-popup');

```

```

3    popup.style.display = 'none';
4    console.log(' 用户确认加购');
5
6    // 准备加购数据
7    const batchData = {
8        line_items: [
9            {
10                product_id: productData.product_id,
11                variant_id: productData.variant_id,
12                quantity: 1
13            }
14        ]
15    };
16
17    // 如果有附加商品，添加到加购数据中
18    if (metaProductId && metaVariantId) {
19        batchData.line_items.push({
20            product_id: metaProductId,
21            variant_id: metaVariantId,
22            quantity: 1
23        });
24    }
25
26    // 调用批量加购 API
27    fetch(window.SHOPLAZZA.routes.root + '/api/cart/batch', {
28        method: 'POST',
29        headers: {
30            'Content-Type': 'application/json'
31        },
32        body: JSON.stringify(batchData)
33    })
34    .then(response => response.json())
35    .then(cartData => {
36        console.log('🛒 批量加购成功:', cartData);
37        alert('商品已成功加入购物车! ');
38        resolve(productData); // 继续加购流程
39    })
40    .catch(error => {
41        console.error('❌ 批量加购失败:', error);
42        alert('加购失败，请重试');
43        reject(new Error('加购失败')); // 中断加购流程
44    });
45 });

```

## 取消加购



代码示例：

- 用户点击“取消”按钮时，隐藏弹窗并中断加购流程。

Code block

```
1 document.getElementById('cancel-add').addEventListener('click', () => {const
  popup = document.getElementById('cart-popup');
2 popup.style.display = 'none';
3 reject(new Error('用户取消加购')); // 中断加购流程});
```

## step4：测试

App测试：<https://www.shoplazza.dev/reference/4test-your-app>

主题测试：<https://www.shoplazza.dev/docs/theme-testing-overview>

## step5：上架审核

<https://www.shoplazza.dev/reference/5submit-app-for-review>

## 其他

### 平台&工具

- 开发者平台：[Partner Center](#)
- Dev Center（插件开发）：[Overview](#)
- Shoplazza CLI：[shoplazza-cli](#)

## FAQ

1. Shoplazza te connect 响应403

```
> pnpm connect

> customeow-app@1.0.0 connect /Users/chenjianhui/Desktop/customeow-shoplazza-plugin
> shoplazza te connect

* Please enter the shoplazza app Client ID: UrSbfMBmiJkWgPCGgVjgUgQhZCbqiStLlrxcd_Ro0V8
* Please enter the shoplazza app Client Secret: YK7UYKmfYCWjXKjGYrNVTGLgjXFt8flWHN3yHi9bkz4
[RESPONSE ERROR] 403 https://partners.shoplazza.com/openapi/2024-07/theme-extensions/connection
[ERROR IN CONNECT] Request failed with status code 403
```

```
🍏 ~/De/customeow-shoplazza-plugin git main !3
```

THEME EXTENSION API 需要开发者提前报备 APP CLIENT ID, 开通白名单