Svelte 应用 API 使用手册

本文档旨在帮助 Svelte 开发者理解并使用后端提供的 Flask API 服务。API 服务划分为几个模块（蓝图）：认证 (Auth)、博客 (Blog)、待办事项 (Todo)、个人看板 (Anchor) 和人工智能 (AI)。

Svelte 集成通用指南

在您的 Svelte 应用中与这些 API 端点交互时，请遵循以下通用指南：

1. 基础 **URL (Base URL)**：
   - 在 Svelte 应用中定义一个 API 请求的基础 URL。例如，在开发环境中可能是 http://localhost:5000/api，在生产环境中则是您的部署域名。
   - 建议将基础 URL 存储在环境变量或配置文件中。
2. **HTTP** 请求：
   - 使用 JavaScript 内置的 fetch API 或如 axios 这样的 HTTP 客户端库来发送请求。
   - 对于 POST, PUT, PATCH 请求，确保将 Content-Type 请求头设置为 application/json（如果API期望接收JSON格式的数据）。
3. 请求体 **(Request Body)**：
   - 对于需要发送数据的请求（如 POST, PUT），通常需要将 JavaScript 对象通过 JSON.stringify() 转换为 JSON 字符串作为请求体。
4. 认证 **(Authentication)**：
   - 许多 API 端点受 JWT (JSON Web Tokens) 保护。
   - 登录后：从登录接口 (/auth/login) 获取 access_token 和 refresh_token。将这些令牌安全地存储起来（例如，使用 Svelte 的 store，并可选择性地持久化到 localStorage 或 sessionStorage）。
   - 发送令牌：对于需要认证的请求，在 Authorization 请求头中附带 access_token，格式为 Bearer <your_access_token>。
   - 令牌刷新：access_token 通常有较短的有效期。当它过期时（API 返回 401 Unauthorized），您需要使用 refresh_token 调用 /auth/refresh 接口来获取新的 access_token。
   - 登出：调用 /auth/logout (撤销 access token) 和 /auth/logout-refresh (撤销 refresh token) 来使令牌失效。
5. 响应处理 **(Response Handling)**：
   - 检查 HTTP 响应状态码。2xx 通常表示成功，4xx 表示客户端错误（如输入无效、未授权），5xx 表示服务器端错误。
   - 解析响应体。大多数 API 会返回 JSON 格式的数据。使用 .json() 方法处理 fetch 的响应。
6. 错误处理 **(Error Handling)**：
   - 在 Svelte 组件中实现健壮的错误处理逻辑。向用户显示有意义的错误信息。

    - 捕获网络错误以及 API 返回的特定错误信息。
7. 加载状态 **(Loading States)**：
    - 在发起 API 请求和等待响应期间，向用户显示加载指示器（如 spinners 或 loading 消息），以改善用户体验。
8. 跨域请求 **(CORS)**：
    - 在开发过程中，Flask 后端需要配置 CORS (Cross-Origin Resource Sharing) 以允许来自 Svelte 开发服务器 (通常是 http://localhost:xxxx，如 http://localhost:5173) 的请求。生产环境中也需要正确配置。

API 模块详解

1. 认证 **API (/api/auth)**

此模块处理用户注册、登录、登出、令牌刷新等认证相关功能。

- 基础路径: /api/auth

1.1 Ping 认证服务

- 用途: 测试认证 API 是否可用。
- 方法: GET
- **URL**: /api/auth/ping
- **Svelte** 示例:

```
async function pingAuth() {
  try {
    const response = await fetch('/api/auth/ping');
    if (!response.ok) {
      throw new Error(`HTTP error! status: ${response.status}`);
    }
    const data = await response.json();
    console.log('Auth API Ping:', data.message); // "Auth API is alive!"
  } catch (error) {
    console.error('Failed to ping Auth API:', error);
  }
}
```

1.2 用户注册

- 用途: 创建一个新用户账户。
- 方法: POST
- **URL**: /api/auth/register
- 请求体 **(JSON)**:

```
{
```

```
  "username": "your_username",
  "email": "user@example.com",
  "password": "your_password"
}
```

- 成功响应 **(201)**:
```
{
  "message": "User registered successfully",
  "user": {
    "id": 1,
    "username": "your_username",
    "email": "user@example.com",
    "created_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ"
  }
}
```

- 错误响应:
  - 400 Bad Request: 缺少字段或数据格式无效。
  - 409 Conflict: 用户名或邮箱已存在。
  - 500 Internal Server Error: 服务器内部错误。

1.3 用户登录

- 用途: 用户登录并获取访问令牌和刷新令牌。
- 方法: POST
- **URL**: /api/auth/login
- 请求体 **(JSON)**:
```
{
  "email": "user@example.com",
  "password": "your_password"
}
```

- 成功响应 **(200)**:
```
{
  "access_token": "your_access_token_here",
  "refresh_token": "your_refresh_token_here"
}
```

- 错误响应:
  - 400 Bad Request: 缺少字段。

- 401 Unauthorized: 邮箱或密码无效。

1.4 用户登出 (撤销 Access Token)

- 用途: 使当前用户的 Access Token 失效。
- 方法: POST
- **URL**: /api/auth/logout
- 认证: 需要有效的 Access Token (@jwt_required())。
- 成功响应 **(200)**:

```
{
  "message": "Access token revoked. User logged out."
}
```

- **Svelte** 示例 **(**假设 **token** 存储在 **store** 中**)**:

```
// authStore.js - Svelte store example
import { writable } from 'svelte/store';
export const accessToken = writable(localStorage.getItem('accessToken'));

// Logout function
async function logout(token) {
  if (!token) return;
  try {
    const response = await fetch('/api/auth/logout', {
      method: 'POST',
      headers: {
        'Authorization': `Bearer ${token}`
      }
    });
    if (response.ok) {
      console.log('Access token revoked.');
      // 清除本地存储的 token
      localStorage.removeItem('accessToken');
      accessToken.set(null);
      // 可能还需要调用 /logout-refresh
    } else {
      const errorData = await response.json();
      console.error('Logout failed:', errorData.error || response.statusText);
    }
  } catch (error) {
    console.error('Error during logout:', error);
```

```
  }
}
```

1.5 用户登出 (撤销 Refresh Token)

- 用途: 使当前用户的 Refresh Token 失效。
- 方法: POST
- **URL**: /api/auth/logout-refresh
- 认证: 需要有效的 Refresh Token (@jwt_required(refresh=True))。
- 成功响应 **(200)**:

```
{
  "message": "Refresh token revoked."
}
```

- 注意: 调用此接口时，Authorization 头应包含 Bearer <your_refresh_token>。

1.6 获取当前用户信息

- 用途: 获取当前已认证用户的个人信息。
- 方法: GET
- **URL**: /api/auth/me
- 认证: 需要有效的 Access Token (@jwt_required())。
- 成功响应 **(200)**:

```
{
  "id": 1,
  "username": "your_username",
  "email": "user@example.com",
  "created_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ",
  "updated_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ"
}
```

1.7 刷新 Access Token

- 用途: 使用有效的 Refresh Token 获取新的 Access Token。
- 方法: POST
- **URL**: /api/auth/refresh
- 认证: 需要有效的 Refresh Token (@jwt_required(refresh=True))。
- 成功响应 **(200)**:

```
{
  "access_token": "new_access_token_here"
}
```

- 注意: 调用此接口时，Authorization 头应包含 Bearer <your_refresh_token>。

1.8 修改密码

- 用途: 修改当前已认证用户的密码。
- 方法: POST
- **URL**: /api/auth/change-password
- 认证: 需要一个“新鲜”的 Access Token (@jwt_required(fresh=True))。用户在登录后立即获得的 Access Token 是新鲜的。如果 Access Token 是通过刷新得到的，则可能不是新鲜的，此时 API 会返回错误，提示用户需要重新登录。
- 请求体 **(JSON)**:

```
{
  "new_password": "your_new_strong_password"
}
```

- 成功响应 **(200)**:

```
{
  "message": "Password updated successfully."
}
```

- 错误响应:
  - 400 Bad Request: 缺少新密码。
  - 401 Unauthorized 或 422 Unprocessable Entity: 如果令牌不是新鲜的 (具体错误取决于 Flask-JWT-Extended 配置)。

2. 博客 **API (/api/blog)**

此模块处理博客文章的 CRUD (创建, 读取, 更新, 删除) 操作。
注意: 根据 blog_bp.py 文件内容，此模块目前使用的是占位符数据 (_posts 字典) 而非数据库。在实际生产中，这些操作会与数据库交互。

- 基础路径: /api/blog

2.1 Ping 博客服务

- 用途: 测试博客 API 是否可用。
- 方法: GET
- **URL**: /api/blog/ping
- 成功响应 **(200)**: {"message": "Blog API is alive!"}

2.2 获取所有博客文章

- 用途: 检索所有博客文章的列表。
- 方法: GET

- **URL**: /api/blog/posts
- 成功响应 **(200)**: (占位符数据示例)
  [
  {"id": 1, "title": "First Post", "content": "This is the content of the first post.", "author_id": 1},
  {"id": 2, "title": "Second Post", "content": "Some interesting thoughts here.", "author_id": 1}
  ]

2.3 获取单篇博客文章

- 用途: 根据 ID 检索单篇博客文章。
- 方法: GET
- **URL**: /api/blog/posts/<post_id> (例如: /api/blog/posts/1)
- 成功响应 **(200)**: (占位符数据示例)
  {"id": 1, "title": "First Post", "content": "This is the content of the first post.", "author_id": 1}

- 错误响应:
  - 404 Not Found: 文章未找到。

2.4 创建新博客文章

- 用途: 创建一篇新的博客文章。
- 方法: POST
- **URL**: /api/blog/posts
- 认证: @jwt_required() (在占位符代码中被注释掉了，但实际应用中应启用)
- 请求体 **(JSON)**:
  {
  "title": "My New Svelte Post",
  "content": "Content about Svelte and APIs..."
  }

- 成功响应 **(201)**: (占位符数据示例)
  {
  "id": 4,
  "title": "My New Svelte Post",
  "content": "Content about Svelte and APIs...",
  "author_id": 1 // 占位符中的 author_id
  }

- 错误响应:
  - 400 Bad Request: 缺少标题或内容。

2.5 更新博客文章

- 用途: 更新一篇已存在的博客文章。
- 方法: PUT
- **URL**: /api/blog/posts/<post_id>
- 认证: @jwt_required() (在占位符代码中被注释掉了，但实际应用中应启用，并检查作者权限)
- 请求体 **(JSON)**:
  ```
  {
    "title": "Updated Svelte Post Title",
    "content": "Updated content..."
  }
  ```
- 成功响应 **(200)**: 更新后的文章对象。
- 错误响应:
  - 400 Bad Request: 缺少标题或内容。
  - 403 Forbidden: 用户无权编辑 (如果启用了作者检查)。
  - 404 Not Found: 文章未找到。

2.6 删除博客文章

- 用途: 删除一篇博客文章。
- 方法: DELETE
- **URL**: /api/blog/posts/<post_id>
- 认证: @jwt_required() (在占位符代码中被注释掉了，但实际应用中应启用，并检查作者权限)
- 成功响应 **(204 No Content)**: 响应体为空。
- 错误响应:
  - 403 Forbidden: 用户无权删除 (如果启用了作者检查)。
  - 404 Not Found: 文章未找到。

3. 待办事项 **API (/api/todo)**

此模块管理用户的待办事项列表。

- 基础路径: /api/todo

3.1 Ping 待办事项服务

- 用途: 测试待办事项 API 是否可用。
- 方法: GET

- **URL**: /api/todo/ping
- 成功响应 **(200)**: {"message": "Todo API is alive!"}

3.2 获取所有待办事项

- 用途: 检索当前认证用户的所有待办事项。
- 方法: GET
- **URL**: /api/todo/todos
- 认证: @jwt_required()
- 成功响应 **(200)**: 待办事项数组。is_current_focus: true 的事项会优先显示，然后按创建时间降序排列。

```
[
 {
  "id": 1,
  "user_id": 1,
  "title": "Finish API documentation",
  "description": "Document all endpoints for Svelte app.",
  "due_date": "2025-12-31", // YYYY-MM-DD 格式或 null
  "status": "in_progress", // 'pending', 'in_progress', 'completed', 'deferred'
  "priority": "high", // 'low', 'medium', 'high'
  "is_current_focus": true,
  "created_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ",
  "updated_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ",
  "completed_at": null // 或 YYYY-MM-DDTHH:MM:SS.ffffffZ
 }
 // ... more todo items
]
```

3.3 创建新待办事项

- 用途: 为当前认证用户创建一个新的待办事项。
- 方法: POST
- **URL**: /api/todo/todos
- 认证: @jwt_required()
- 请求体 **(JSON)**:

```
{
 "title": "Learn SvelteKit",
 "description": "Go through the official SvelteKit tutorial.", // 可选
 "due_date": "2025-06-15", // 可选, YYYY-MM-DD
 "status": "pending", // 可选, 默认 'pending'
```

```
  "priority": "medium" // 可选, 默认 'medium'
}
```

    - is_current_focus 默认为 false，通常通过 PUT 请求更新。
- 成功响应 **(201)**: 新创建的待办事项对象。
- 错误响应:
    - 400 Bad Request: 字段无效（如标题为空、日期格式错误、状态或优先级无效）。

3.4 获取单个待办事项

- 用途: 根据 ID 检索单个待办事项。
- 方法: GET
- **URL**: /api/todo/todos/<todo_id>
- 认证: @jwt_required()
- 成功响应 **(200)**: 待办事项对象。
- 错误响应:
    - 403 Forbidden: 无权访问该事项。
    - 404 Not Found: 事项未找到。

3.5 更新待办事项

- 用途: 更新一个已存在的待办事项。可以更新标准字段以及 is_current_focus 标志。
- 方法: PUT
- **URL**: /api/todo/todos/<todo_id>
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。

```
{
  "title": "Learn SvelteKit Advanced", // 可选
  "status": "completed", // 可选
  "is_current_focus": false // 可选
  // ... 其他可更新字段
}
```

    - 如果 status 更新为 completed，completed_at 会自动设置。如果从 completed 改为其他状态，completed_at 会设为 null。
- 成功响应 **(200)**: 更新后的待办事项对象。
- 错误响应:
    - 400 Bad Request: 字段无效。
    - 403 Forbidden: 无权更新该事项。
    - 404 Not Found: 事项未找到。

3.6 删除待办事项

- 用途: 删除一个待办事项。
- 方法: DELETE
- **URL**: /api/todo/todos/<todo_id>
- 认证: @jwt_required()
- 成功响应 **(204 No Content)**: 响应体为空。
- 错误响应:
  - 403 Forbidden: 无权删除该事项。
  - 404 Not Found: 事项未找到。

4. 个人看板 **API (/api/anchor)**

此模块用于管理用户的"个人看板"信息，包括个人资料、成就（做过什么）和未来计划（打算做什么）。"当前焦点"（正在做什么）已整合到待办事项的 is_current_focus 字段中。

- 基础路径: /api/anchor

4.1 Ping 看板服务

- 用途: 测试看板 API 是否可用。
- 方法: GET
- **URL**: /api/anchor/ping
- 成功响应 **(200)**: {"message": "Anchor API is alive!"}

4.2 用户资料 **(User Profile)**

4.2.1 获取用户看板资料

- 用途: 获取当前认证用户的看板资料。如果用户尚无资料，会自动创建一条空的资料记录。
- 方法: GET
- **URL**: /api/anchor/profile
- 认证: @jwt_required()
- 成功响应 **(200)**: 用户资料对象。

```
{
  "id": 1, // user_id
  "professional_title": "软件工程师", // 可为 null
  "one_liner_bio": "热衷于构建可扩展的 Web 应用。", // 可为 null
  "skill": "Svelte, Python, Flask", // 可为 null
  "summary": "详细的个人总结...", // 可为 null
  "created_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ",
  "updated_at": "YYYY-MM-DDTHH:MM:SS.ffffffZ"
}
```

4.2.2 更新用户看板资料

- 用途: 更新当前认证用户的看板资料。
- 方法: PUT
- **URL**: /api/anchor/profile
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。

```
{
  "professional_title": "高级软件工程师", // 可选
  "one_liner_bio": "更新的简介。" // 可选
  // "skill": ..., "summary": ...
}
```

- 成功响应 **(200)**: 更新后的用户资料对象。
- 错误响应:
  - 400 Bad Request: 请求体为空或字段类型错误。

4.3 成就 **(Achievements - "做过什么")**

4.3.1 创建成就

- 用途: 为当前用户添加一条新的成就记录。
- 方法: POST
- **URL**: /api/anchor/achievements
- 认证: @jwt_required()
- 请求体 **(JSON)**:

```
{
  "title": "项目A成功上线", // 必填, 非空字符串
  "description": "负责项目A的前后端开发...", // 可选, 字符串
  "quantifiable_results": "用户增长20%", // 可选, 字符串
  "core_skills_json": ["Svelte", "API设计", "项目管理"], // 可选, 字符串列表
  "date_achieved": "2024-10-15" // 可选, YYYY-MM-DD
}
```

- 成功响应 **(201)**: 新创建的成就对象。
- 错误响应:
  - 400 Bad Request: 字段无效（如标题为空、core_skills_json 不是列表、日期格式错误）。

4.3.2 获取所有成就

- 用途: 获取当前用户的所有成就记录，按完成日期降序、创建时间降序排列。
- 方法: GET

- **URL**: /api/anchor/achievements
- 认证: @jwt_required()
- 成功响应 **(200)**: 成就对象数组。

4.3.3 获取单个成就

- 用途: 根据 ID 获取单个成就记录。
- 方法: GET
- **URL**: /api/anchor/achievements/<achievement_id>
- 认证: @jwt_required()
- 成功响应 **(200)**: 成就对象。
- 错误响应:
  - 403 Forbidden: 无权访问。
  - 404 Not Found: 未找到。

4.3.4 更新成就

- 用途: 更新指定的成就记录。
- 方法: PUT
- **URL**: /api/anchor/achievements/<achievement_id>
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。
- 成功响应 **(200)**: 更新后的成就对象。
- 错误响应:
  - 400 Bad Request: 字段无效。
  - 403 Forbidden: 无权更新。
  - 404 Not Found: 未找到。

4.3.5 删除成就

- 用途: 删除指定的成就记录。
- 方法: DELETE
- **URL**: /api/anchor/achievements/<achievement_id>
- 认证: @jwt_required()
- 成功响应 **(204 No Content)**: 响应体为空。
- 错误响应:
  - 403 Forbidden: 无权删除。
  - 404 Not Found: 未找到。

4.4 未来计划 **(Future Plans - "打算做什么")**

4.4.1 创建未来计划

- 用途: 为当前用户添加一个新的未来计划。
- 方法: POST

- **URL**: /api/anchor/future_plans
- 认证: @jwt_required()
- 请求体 **(JSON)**:
  {
  "title": "学习AI技术", // 必填, 非空字符串
  "description": "系统学习机器学习和深度学习基础。", // 必填, 非空字符串
  "goal_type": "学习提升", // 可选, 字符串, 最多50字符
  "status": "active", // 可选, 默认 'active'。允许值: 'active', 'achieved', 'deferred', 'abandoned'
  "target_date": "2026-12-31" // 可选, YYYY-MM-DD
  }
- 成功响应 **(201)**: 新创建的未来计划对象。
- 错误响应:
  - 400 Bad Request: 字段无效。

4.4.2 获取所有未来计划

- 用途: 获取当前用户的所有未来计划，按目标日期升序、创建时间降序排列。
- 方法: GET
- **URL**: /api/anchor/future_plans
- 认证: @jwt_required()
- 成功响应 **(200)**: 未来计划对象数组。

4.4.3 获取单个未来计划

- 用途: 根据 ID 获取单个未来计划。
- 方法: GET
- **URL**: /api/anchor/future_plans/<plan_id>
- 认证: @jwt_required()
- 成功响应 **(200)**: 未来计划对象。
- 错误响应:
  - 403 Forbidden: 无权访问。
  - 404 Not Found: 未找到。

4.4.4 更新未来计划

- 用途: 更新指定的未来计划。
- 方法: PUT
- **URL**: /api/anchor/future_plans/<plan_id>
- 认证: @jwt_required()
- 请求体 **(JSON)**: 包含要更新的字段。
- 成功响应 **(200)**: 更新后的未来计划对象。

- 错误响应:
  - 400 Bad Request: 字段无效。
  - 403 Forbidden: 无权更新。
  - 404 Not Found: 未找到。

4.4.5 删除未来计划

- 用途: 删除指定的未来计划。
- 方法: DELETE
- **URL**: /api/anchor/future_plans/<plan_id>
- 认证: @jwt_required()
- 成功响应 **(204 No Content)**: 响应体为空。
- 错误响应:
  - 403 Forbidden: 无权删除。
  - 404 Not Found: 未找到。

5. AI API (/api/ai)

此模块用于与外部 AI 服务集成，例如文本生成。
注意: 此模块的实际功能依赖于外部 AI 服务（如 OpenAI）的配置和 API 密钥。占位符代码中，API 密钥部分被注释。

- 基础路径: /api/ai

5.1 Ping AI 服务

- 用途: 测试 AI API 是否可用。
- 方法: GET
- **URL**: /api/ai/ping
- 成功响应 **(200)**: {"message": "AI API is alive!"}

5.2 生成文本

- 用途: 与外部 AI 文本生成服务交互。
- 方法: POST
- **URL**: /api/ai/generate-text
- 认证: @jwt_required() (在占位符代码中被注释掉了，但如果 AI 使用是用户特定或计量的，则应启用)
- 请求体 **(JSON)**:
  ```
  {
    "prompt": "请用 Svelte 写一个计数器组件的例子"
  }
  ```
- 成功响应 **(200)**:
  ```
  {
  ```

"generated_text": "这是AI生成的文本内容..."
// 或占位符响应: "Placeholder response for prompt: '请用 Svelte 写一个计数器组件的例子...'"
}

- 错误响应:
  - 400 Bad Request: 缺少 prompt 字段。
  - 500 Internal Server Error: 调用 AI 服务失败或认证失败。
  - 503 Service Unavailable: AI 服务未配置 (例如，缺少 API 密钥)。

Svelte 代码示例片段

以下是如何在 Svelte 组件中进行 API 调用的通用模式：

```
<script>
  import { onMount } from 'svelte';
  import { accessToken } from './authStore'; // 假设您有一个 authStore

  let data = null;
  let error = null;
  let loading = false;
  const API_BASE_URL = 'http://localhost:5000/api'; // 或者从环境变量读取

  async function fetchData(endpoint, method = 'GET', body = null, requiresAuth = false) {
    loading = true;
    error = null;
    try {
      const headers = {
        'Content-Type': 'application/json',
      };
      if (requiresAuth) {
        const token = localStorage.getItem('accessToken'); // 或者从 store 获取
        if (!token) {
          throw new Error('No access token found. Please log in.');
        }
        headers['Authorization'] = `Bearer ${token}`;
      }

      const options = {
```

```
      method,
      headers,
    };

    if (body && (method === 'POST' || method === 'PUT')) {
      options.body = JSON.stringify(body);
    }

    const response = await fetch(`${API_BASE_URL}${endpoint}`, options);

    if (!response.ok) {
      const errorData = await response.json().catch(() => ({ message:
response.statusText }));
      throw new Error(errorData.error || errorData.message || `HTTP error! status:
${response.status}`);
    }

    if (response.status === 204) { // No Content
      return null;
    }
    return await response.json();

  } catch (err) {
    error = err.message;
    console.error(`Failed to ${method} ${endpoint}:`, err);
    return null; // 或抛出错误供上层处理
  } finally {
    loading = false;
  }
 }

 async function getMyTodos() {
  data = await fetchData('/todo/todos', 'GET', null, true);
 }

 async function createNewTodo() {
  const newTodoData = { title: "My new Svelte task", priority: "high" };
  const result = await fetchData('/todo/todos', 'POST', newTodoData, true);
  if (result) {
```

```
    // 成功创建，可以刷新列表或将新项添加到现有数据中
    getMyTodos();
   }
  }

  onMount(() => {
   // 示例：如果用户已登录，则加载待办事项
   if (localStorage.getItem('accessToken')) {
    getMyTodos();
   }
  });
</script>

{#if loading}
  <p>正在加载...</p>
{/if}

{#if error}
  <p style="color: red;">错误: {error}</p>
{/if}

{#if data}
  <pre>{JSON.stringify(data, null, 2)}</pre>
{/if}

<button on:click={createNewTodo} disabled={loading}>创建新待办</button>
<button on:click={getMyTodos} disabled={loading}>刷新待办列表</button>
```

请根据您的具体需求调整上述代码。确保在 Svelte 应用中妥善管理认证令牌、用户状态和错误信息。