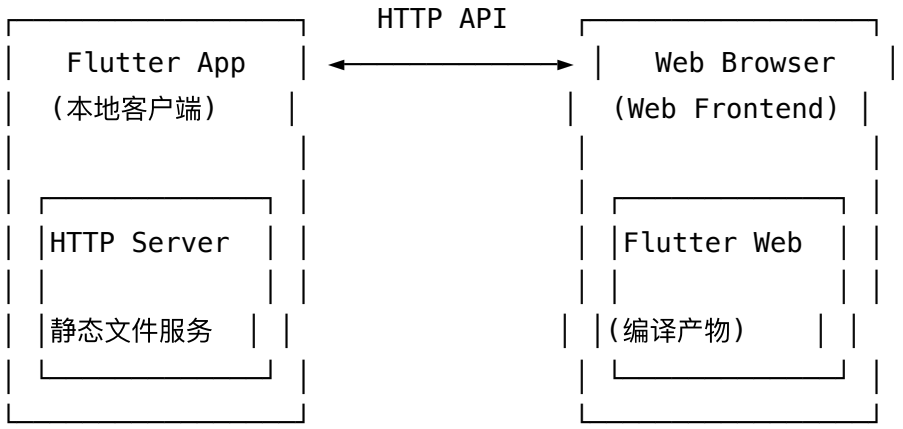
NipaPlay Web服务器实现方案

概述

本文档描述如何在NipaPlay Flutter应用中实现内置Web服务器功能，使用户可以通过浏览器访问应用的Web版本，实现与本地客户端相同的功能体验。

技术方案

1. 架构设计



2. 核心组件

2.1 HTTP服务器

- 使用Dart的 `shelf` 包创建内置HTTP服务器
- 提供静态文件服务（Flutter Web编译产物）
- 提供REST API接口用于数据交互

2.2 Web前端

- 编译Flutter Web版本作为静态资源
- 通过HTTP API与本地客户端通信
- 实现与桌面版相同的功能界面

2.3 数据同步

- 共享相同的数据库和配置文件
- 实时同步播放状态、媒体库信息等

实现步骤

第一阶段：基础Web服务器

1. 添加依赖

在 `pubspec.yaml` 中添加：

```yaml
dependencies:
  shelf: ^1.4.1
  shelf_static: ^1.1.2
  shelf_router: ^1.1.4
  shelf_cors_headers: ^0.1.5
```

2. 创建Web服务器服务类

```dart
// lib/services/web_server_service.dart
class WebServerService {
  static const String _enabledKey = 'web_server_enabled';
  static const String _portKey = 'web_server_port';

  HttpServer? _server;
  int _port = 8080;
  bool _isRunning = false;

  // 启动服务器
  Future<bool> startServer() async;

  // 停止服务器
  Future<void> stopServer() async;

  // 获取本地访问地址
  List<String> getAccessUrls();

  // 设置自动启动
  Future<void> setAutoStart(bool enabled) async;
}
```

3. 在设置页面添加Web服务器配置

在 `lib/pages/settings/general_page.dart` 中添加Web服务器设置项：

```
// Web服务器设置区域
Card(
  child: Column(
    children: [
      SwitchListTile(
        title: Text("启用Web服务器"),
        subtitle: Text("允许通过浏览器访问应用"),
        value: _webServerEnabled,
        onChanged: _toggleWebServer,
      ),
      if (_webServerEnabled) ...[
        ListTile(
          title: Text("访问地址"),
          subtitle: Column(
            crossAxisAlignment: CrossAxisAlignment.start,
            children: _getAccessUrls().map((url) =>
              Text(url, style: TextStyle(fontFamily: 'monospace'))
            ).toList(),
          ),
          trailing: IconButton(
            icon: Icon(Icons.copy),
            onPressed: _copyUrls,
          ),
        ),
        ListTile(
          title: Text("端口设置"),
          subtitle: Text("当前端口: $_currentPort"),
          trailing: IconButton(
            icon: Icon(Icons.edit),
            onPressed: _showPortDialog,
          ),
        ),
      ],
    ],
  ),
)
```

第二阶段：API接口设计

1. 媒体库API

```
// GET /api/media/libraries - 获取媒体库列表
// GET /api/media/library/{id}/items - 获取媒体库内容
// GET /api/media/item/{id} - 获取媒体项详情
// POST /api/media/scan - 触发媒体库扫描
```

2. 播放控制API

```
// POST /api/player/play - 播放媒体
// POST /api/player/pause - 暂停播放
// POST /api/player/seek - 跳转播放位置
// GET /api/player/status - 获取播放状态
// POST /api/player/volume - 设置音量
```

3. 设置API

```
// GET /api/settings - 获取所有设置
// PUT /api/settings/{key} - 更新设置项
// GET /api/settings/theme - 获取主题设置
```

4. 弹幕API

```
// GET /api/danmaku/{episodeId} - 获取弹幕数据
// POST /api/danmaku/{episodeId} - 发送弹幕
// GET /api/danmaku/search - 搜索弹幕
```

第三阶段：Web前端适配

1. 编译Web版本

```
flutter build web --release --web-renderer canvaskit
```

2. 创建构建脚本

```
#!/bin/bash
# build_web_assets.sh

echo "正在编译Web版本..."
flutter build web --release --web-renderer canvaskit

echo "正在复制Web资源到assets目录..."
mkdir -p assets/web
cp -r build/web/* assets/web/

echo "Web资源构建完成!"
```

3. Web版本特殊处理

```
// lib/utils/platform_helper.dart
class PlatformHelper {
  static bool get isWebRemote =>
    kIsWeb && Uri.base.host != 'localhost' && Uri.base.host != '127.0.0.1';

  static String get apiBaseUrl =>
    isWebRemote ? 'http://${Uri.base.host}:${Uri.base.port}' : '';
}
```

第四阶段：数据同步机制

1. 实时数据同步

```dart
// lib/services/web_sync_service.dart
class WebSyncService {
  Timer? _syncTimer;

  void startSync() {
    _syncTimer = Timer.periodic(Duration(seconds: 1), (_) {
      _syncPlaybackState();
      _syncMediaLibrary();
    });
  }

  Future<void> _syncPlaybackState() async {
    // 同步播放状态到Web端
  }

  Future<void> _syncMediaLibrary() async {
    // 同步媒体库变更到Web端
  }
}
```

2. WebSocket连接（可选）

```dart
// 用于实时双向通信
class WebSocketService {
  WebSocketChannel? _channel;

  void connect() {
    _channel = WebSocketChannel.connect(
      Uri.parse('ws://localhost:$_port/ws')
    );
  }

  void sendMessage(Map<String, dynamic> message) {
    _channel?.sink.add(jsonEncode(message));
  }
}
```

安全考虑

1. 访问控制

```
// 可选：添加简单的token认证
class AuthService {
  static String _generateToken() =>
    DateTime.now().millisecondsSinceEpoch.toString();

  static bool validateToken(String token) {
    // 简单的时效性检查
    final timestamp = int.tryParse(token) ?? 0;
    final now = DateTime.now().millisecondsSinceEpoch;
    return (now - timestamp) < Duration(hours: 24).inMilliseconds;
  }
}
```

2. 跨域处理

```
// 在shelf服务器中添加CORS headers
final handler = Pipeline()
  .addMiddleware(corsHeaders())
  .addMiddleware(logRequests())
  .addHandler(router);
```

3. 本地网络限制

```
// 只允许本地网络访问
bool isLocalNetwork(String clientIp) {
  return clientIp.startsWith('192.168.') ||
         clientIp.startsWith('10.') ||
         clientIp.startsWith('172.') ||
         clientIp == '127.0.0.1';
}
```

用户体验优化

1. 自动发现

- 在应用启动时显示二维码，便于移动设备扫码访问
- 支持局域网设备自动发现

2. 响应式设计

- Web版本自动适配不同屏幕尺寸
- 移动端优化的触控界面

3. 离线缓存

- 使用Service Worker缓存静态资源
- 支持离线访问基本功能

文件结构

```
lib/
├── services/
│   ├── web_server_service.dart     # Web服务器核心服务
│   ├── web_api_service.dart        # API接口实现
│   ├── web_sync_service.dart       # 数据同步服务
│   └── web_socket_service.dart     # WebSocket服务（可选）
├── controllers/
│   ├── media_controller.dart       # 媒体API控制器
│   ├── player_controller.dart      # 播放API控制器
│   └── settings_controller.dart    # 设置API控制器
├── utils/
│   ├── web_server_utils.dart       # Web服务器工具函数
│   └── network_utils.dart          # 网络工具函数
└── widgets/
    ├── web_server_status.dart      # Web服务器状态组件
    └── qr_code_display.dart        # 二维码显示组件
```

配置示例

默认配置

```
class WebServerConfig {
  static const int defaultPort = 8080;
  static const bool defaultAutoStart = false;
  static const String webAssetsPath = 'assets/web';
  static const Duration syncInterval = Duration(seconds: 1);
}
```

使用流程

1. **启用Web服务器**
   - 在设置->通用设置中开启Web服务器功能
   - 设置端口号（默认8080）
2. **获取访问地址**
   - 应用自动显示所有可用的访问地址
   - 扫描二维码快速访问
3. **浏览器访问**
   - 在同一局域网的任意设备上打开浏览器
   - 访问显示的地址即可使用Web版本
4. **功能同步**
   - 播放状态实时同步
   - 媒体库变更自动更新
   - 设置修改即时生效

技术优势

1. **统一体验**：Web版本与桌面版功能完全一致
2. **无需安装**：任意设备浏览器即可访问
3. **实时同步**：所有操作实时同步到本地客户端
4. **跨平台**：支持所有现代浏览器和设备
5. **安全可控**：仅限局域网访问，用户完全掌控

扩展可能

1. **远程访问**：通过VPN或端口转发实现远程访问
2. **多用户支持**：支持多个用户同时访问
3. **API开放**：为第三方应用提供API接口
4. **插件系统**：支持Web端插件扩展

这个方案将让NipaPlay成为一个真正的多端统一媒体中心！