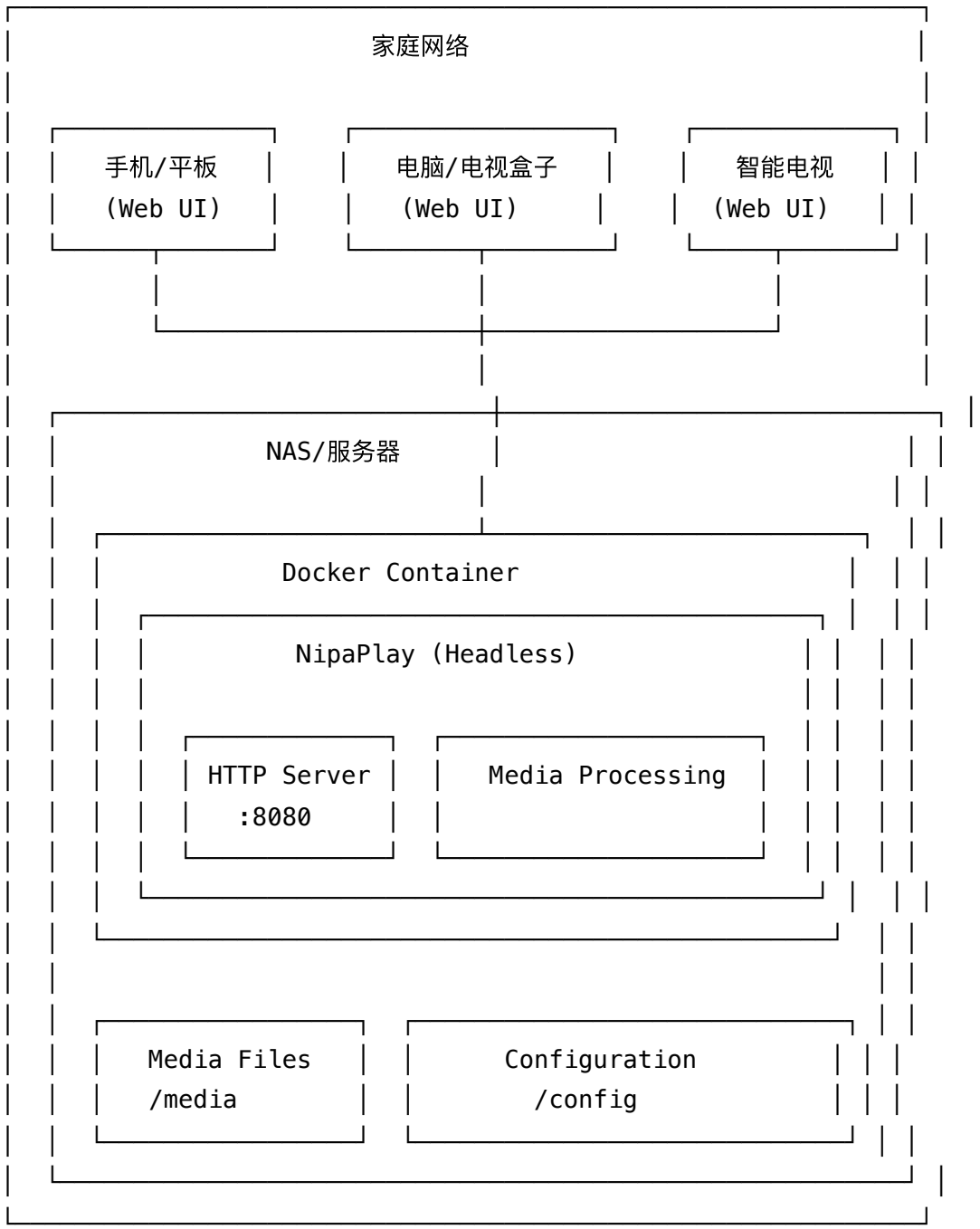
NipaPlay 容器化部署指南

概述

本指南描述如何将NipaPlay打包为Docker容器，并部署到NAS、VPS或家庭服务器上，实现7x24小时运行的家庭媒体中心。

部署架构


家庭网络
手机/平板
(Web UI)
电脑/电视盒子
(Web UI)
智能电视
(Web UI)
NAS/服务器
Docker Container
NipaPlay (Headless)
HTTP Server
:8080
Media Processing
Media Files
/media
Configuration
/config

Docker化方案

1. 创建无头模式支持

修改main.dart支持headless模式

```
// lib/main.dart
void main(List<String> args) async {
  WidgetsFlutterBinding.ensureInitialized();

  // 检查是否为无头模式
  final bool isHeadless = args.contains('--headless') ||
                          Platform.environment['NIPAPLAY_HEADLESS'] == 'true';

  if (isHeadless) {
    await runHeadlessMode(args);
  } else {
    await runGuiMode(args);
  }
}

Future<void> runHeadlessMode(List<String> args) async {
  print('🚀 NipaPlay正在以无头模式启动...');

  // 初始化核心服务
  await initializeCoreServices();

  // 启动Web服务器
  final webServer = WebServerService();
  await webServer.startServer();

  print('✅ NipaPlay Web服务器已启动');
  print('🌐 访问地址: ${webServer.getAccessUrls().join(', ')}');

  // 保持运行
  await Future.delayed(Duration(days: 365)); // 持续运行
}

Future<void> runGuiMode(List<String> args) async {
  // 原有的GUI启动逻辑
```

```
  runApp(MyApp());
}
```

创建核心服务初始化器

```
// lib/services/headless_initializer.dart
class HeadlessInitializer {
  static Future<void> initializeCoreServices() async {
    // 初始化数据库
    await DatabaseService.initialize();

    // 初始化设置存储
    await SettingsStorage.initialize();

    // 初始化媒体库服务
    await MediaLibraryService.initialize();

    // 初始化弹幕服务
    await DanmakuService.initialize();

    // 启动后台任务
    await BackgroundTaskService.start();

    print('✅ 核心服务初始化完成');
  }
}
```

2. 创建Dockerfile

```
# Dockerfile
FROM ubuntu:22.04

# 安装系统依赖
RUN apt-get update && apt-get install -y \
    curl \
    git \
    unzip \
    xz-utils \
    zip \
    libglu1-mesa \
    libc6-dev \
    libstdc++6 \
    libgcc1 \
    libatomic1 \
    fonts-noto-cjk \
    ca-certificates \
    && rm -rf /var/lib/apt/lists/*

# 安装Flutter
RUN git clone https://github.com/flutter/flutter.git -b stable --depth 1 /flutter
ENV PATH="/flutter/bin:${PATH}"

# 设置工作目录
WORKDIR /app

# 复制项目文件
COPY . .

# 构建应用
RUN flutter doctor
RUN flutter pub get
RUN flutter build linux --release

# 构建Web版本
RUN flutter build web --release --web-renderer canvaskit
RUN mkdir -p build/linux/x64/release/bundle/data/flutter_assets/web
RUN cp -r build/web/* build/linux/x64/release/bundle/data/flutter_assets/web/

# 设置运行时环境
ENV NIPAPLAY_HEADLESS=true
```

```
ENV NIPAPLAY_WEB_PORT=8080
ENV NIPAPLAY_DATA_DIR=/data
ENV NIPAPLAY_MEDIA_DIR=/media

# 创建数据目录
RUN mkdir -p /data /media

# 暴露端口
EXPOSE 8080

# 设置启动命令
CMD ["./build/linux/x64/release/bundle/nipaplay", "--headless"]
```

3. 创建Docker Compose配置

```yaml
# docker-compose.yml
version: '3.8'

services:
  nipaplay:
    build: .
    container_name: nipaplay-server
    restart: unless-stopped
    ports:
      - "8080:8080"
    volumes:
      # 媒体文件目录
      - /path/to/your/media:/media:ro
      # 配置和数据目录
      - ./data:/data
      # 缓存目录
      - ./cache:/app/cache
    environment:
      - NIPAPLAY_HEADLESS=true
      - NIPAPLAY_WEB_PORT=8080
      - NIPAPLAY_AUTO_SCAN=true
      - NIPAPLAY_LOG_LEVEL=INFO
    networks:
      - nipaplay-net

networks:
  nipaplay-net:
    driver: bridge
```

4. 构建和部署脚本

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

echo "🚀 开始构建NipaPlay Docker镜像..."

# 构建镜像
docker build -t nipaplay:latest .

# 检查构建结果
if [ $? -eq 0 ]; then
    echo "✅ 镜像构建成功"
else
    echo "❌ 镜像构建失败"
    exit 1
fi

# 创建数据目录
mkdir -p ./data ./cache

# 启动服务
echo "🚀 启动NipaPlay服务..."
docker-compose up -d

# 显示状态
echo "📊 服务状态:"
docker-compose ps

echo "🌐 访问地址: http://$(hostname -I | awk '{print $1}'):8080"
echo "📁 请将媒体文件放置在绑定的媒体目录中"
```

NAS部署方案

1. 群晖NAS (Synology)

使用Docker套件

```
# 1. 在套件中心安装Docker
# 2. 上传镜像或从仓库拉取
# 3. 创建容器时配置:

# 端口设置
本地端口: 8080 -> 容器端口: 8080

# 卷映射
/volume1/media -> /media (只读)
/volume1/docker/nipaplay/data -> /data
/volume1/docker/nipaplay/cache -> /app/cache

# 环境变量
NIPAPLAY_HEADLESS=true
NIPAPLAY_AUTO_SCAN=true
```

任务计划设置

```
# 在控制面板 -> 任务计划中添加
# 类型: 用户定义的脚本
# 计划: 开机时运行

#!/bin/bash
docker start nipaplay-server
```

2. 威联通NAS (QNAP)

Container Station配置

```
# 在Container Station中创建应用
名称: NipaPlay
镜像: nipaplay:latest
网络模式: bridge
端口: 8080:8080

# 共享文件夹映射
/share/Multimedia -> /media
/share/Container/nipaplay -> /data
```

3. OpenMediaVault

```
# 1. 安装Docker插件
omv-extras-org

# 2. 创建docker-compose.yml
# 3. 使用Portainer管理容器
```

配置优化

1. 环境变量配置

```
# .env 文件
NIPAPLAY_HEADLESS=true
NIPAPLAY_WEB_PORT=8080
NIPAPLAY_DATA_DIR=/data
NIPAPLAY_MEDIA_DIR=/media
NIPAPLAY_CACHE_DIR=/cache

# 媒体库设置
NIPAPLAY_AUTO_SCAN=true
NIPAPLAY_SCAN_INTERVAL=3600  # 1小时扫描一次

# 日志设置
NIPAPLAY_LOG_LEVEL=INFO
NIPAPLAY_LOG_FILE=/data/logs/nipaplay.log

# 网络设置
NIPAPLAY_CORS_ORIGINS=*
NIPAPLAY_MAX_CONNECTIONS=100
```

2. 性能优化配置

```dart
// lib/config/headless_config.dart
class HeadlessConfig {
  // 内存使用优化
  static const int maxCacheSize = 512; // MB
  static const int maxConcurrentStreams = 10;

  // 扫描优化
  static const Duration scanInterval = Duration(hours: 1);
  static const int maxScanThreads = 4;

  // 网络优化
  static const int httpTimeout = 30; // 秒
  static const int maxHttpConnections = 50;

  // 数据库优化
  static const int dbConnectionPool = 5;
  static const Duration dbTimeout = Duration(seconds: 10);
}
```

监控和维护

1. 健康检查

```
# 在Dockerfile中添加健康检查
HEALTHCHECK --interval=30s --timeout=10s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:8080/api/health || exit 1
```

```dart
// 添加健康检查端点
// lib/controllers/health_controller.dart
class HealthController {
  static Response healthCheck(Request request) {
    return Response.ok(jsonEncode({
      'status': 'healthy',
      'timestamp': DateTime.now().toIso8601String(),
      'version': AppInfo.version,
      'uptime': _getUptime(),
      'services': {
        'database': DatabaseService.isHealthy,
        'media_scanner': ScanService.isRunning,
        'web_server': true,
      }
    }));
  }
}
```

2. 日志管理

```dart
// lib/services/logging_service.dart
class LoggingService {
  static void setupHeadlessLogging() {
    final logFile = File('${Platform.environment['NIPAPLAY_DATA_DIR']}/logs/app.log');

    // 设置日志轮转
    Logger.root.onRecord.listen((record) {
      final logEntry = '${record.time}: ${record.level.name}: ${record.message}\n';
      logFile.writeAsStringSync(logEntry, mode: FileMode.append);

      // 控制台输出
      print(logEntry.trim());
    });
  }
}
```

3. 备份脚本

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

BACKUP_DIR="/path/to/backup"
DATA_DIR="/path/to/nipaplay/data"
DATE=$(date +%Y%m%d_%H%M%S)

echo "🔄 开始备份NipaPlay数据..."

# 创建备份目录
mkdir -p "$BACKUP_DIR"

# 压缩数据目录
tar -czf "$BACKUP_DIR/nipaplay_backup_$DATE.tar.gz" -C "$DATA_DIR" .

# 保留最近7天的备份
find "$BACKUP_DIR" -name "nipaplay_backup_*.tar.gz" -mtime +7 -delete

echo "✅ 备份完成: nipaplay_backup_$DATE.tar.gz"
```

使用场景

1. 家庭媒体服务器

- **部署位置**: 家中NAS或迷你主机
- **访问方式**: 局域网内所有设备
- **存储**: 本地大容量硬盘

2. VPS云服务器

- **部署位置**: 阿里云、腾讯云等
- **访问方式**: 通过域名访问
- **存储**: 云盘或对象存储

3. 办公室娱乐系统

- **部署位置**: 办公室服务器

- **访问方式**: 内网访问
- **内容**: 教育视频、娱乐内容

优势总结

1. 部署优势

- **一次构建，到处运行** - Docker保证环境一致性
- **资源占用低** - 无头模式下内存占用极小
- **24x7运行** - 适合服务器环境长期运行

2. 管理优势

- **远程管理** - 通过Web界面管理所有功能
- **自动更新** - 支持容器热更新
- **集中存储** - 媒体文件统一管理

3. 扩展优势

- **负载均衡** - 可部署多实例分担负载
- **API开放** - 为其他应用提供接口
- **插件系统** - 支持功能扩展

这样NipaPlay就从一个桌面应用进化成了真正的**企业级媒体服务器解决方案**！🚀