# Dash Duck 软件配置与运维文档

## 1. 配置管理

### 1.1 环境配置

```yaml
# pubspec.yaml - 项目配置
name: dash_duck
description: A Flutter application for SYSU course schedule
version: 1.0.0+1

environment:
  sdk: '>=3.0.0 <4.0.0'
  flutter: ">=3.0.0"

dependencies:
  flutter:
    sdk: flutter
  cupertino_icons: ^1.0.2
  # home_widget: ^0.4.0  # 计划添加

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^3.0.0
```

### 1.2 构建配置

```yaml
# analysis_options.yaml - 代码质量配置
include: package:flutter_lints/flutter.yaml

linter:
  rules:
    prefer_const_constructors: true
    prefer_const_literals_to_create_immutables: true
    avoid_unnecessary_containers: true
```

## 2. 版本控制策略

### 2.1 分支管理

```
master (主分支)
├── develop (开发分支)
```

```
├── feature/schedule-ui (功能分支)
├── feature/grade-system (功能分支)
└── hotfix/location-parser (修复分支)
```

## 2.2 版本命名规范

- **主版本号**: 不兼容的API修改
- **次版本号**: 向下兼容的功能性新增
- **修订号**: 向下兼容的问题修正
- **构建号**: 每次构建递增

# 3. 持续集成配置

## 3.1 CI/CD流水线

```yaml
# .github/workflows/main.yml
name: Flutter CI

on:
  push:
    branches: [ master, develop ]
  pull_request:
    branches: [ master ]

jobs:
  test:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout code
      uses: actions/checkout@v3

    - name: Setup Flutter
      uses: subosito/flutter-action@v2
      with:
        flutter-version: '3.16.0'

    - name: Install dependencies
      run: flutter pub get

    - name: Run tests
      run: flutter test --coverage

    - name: Analyze code
      run: flutter analyze

    - name: Build APK
```

```
      run: flutter build apk --release

    - name: Upload artifacts
      uses: actions/upload-artifact@v3
      with:
        name: release-apk
        path: build/app/outputs/flutter-apk/app-release.apk
```

### 3.2 质量门禁

- 代码覆盖率 > 80%
- 静态分析无错误
- 所有测试用例通过
- 代码审查通过

## 4. 部署策略

### 4.1 多环境部署

```
开发环境 (Development)
├── 自动部署: 每次commit
├── 测试数据: 模拟数据
└── 调试模式: 开启

测试环境 (Staging)
├── 手动部署: develop分支
├── 真实数据: 测试数据
└── 性能监控: 开启

生产环境 (Production)
├── 审批部署: master分支
├── 生产数据: 真实数据
└── 错误追踪: 开启
```

### 4.2 发布策略

- **蓝绿部署**: 零停机更新
- **灰度发布**: 逐步推广新版本
- **回滚机制**: 快速恢复旧版本

## 5. 运维监控

### 5.1 应用监控

```
// 错误监控示例
void main() {
  FlutterError.onError = (FlutterErrorDetails details) {
    // 上报错误到监控平台
    FirebaseCrashlytics.instance.recordFlutterError(details);
  };

  runZonedGuarded(
    () => runApp(DashDuck()),
    (error, stackTrace) {
      // 捕获未处理的异步错误
      FirebaseCrashlytics.instance.recordError(error, stackTrace);
    },
  );
}
```

## 5.2 性能监控

- **启动时间**: 应用冷启动耗时
- **内存使用**: 内存泄漏检测
- **网络请求**: API响应时间
- **用户行为**: 页面访问统计

## 5.3 日志管理

```
// 日志配置
class Logger {
  static void info(String message) {
    if (kDebugMode) {
      print('[INFO] ${DateTime.now()}: $message');
    }
  }

  static void error(String message, [dynamic error]) {
    print('[ERROR] ${DateTime.now()}: $message');
    if (error != null) {
      FirebaseCrashlytics.instance.log(message);
    }
  }
}
```

# 6. 安全配置

## 6.1 数据安全

- **传输加密**: HTTPS协议

- **存储加密**: 敏感数据加密存储
- **权限控制**: 最小权限原则

## 6.2 代码安全

- **依赖检查**: 定期更新依赖包
- **代码混淆**: 发布版本代码混淆
- **证书管理**: 签名证书安全存储