

我为您设计一个完整的数据手套系统项目结构，涵盖从硬件到移动端的全栈解决方案。

项目整体文件结构

```
DataGloveSystem/
├── README.md
├── docs/                                # 项目文档
│   ├── API_Documentation.md
│   ├── Hardware_Setup.md
│   ├── Deployment_Guide.md
│   └── Protocol_Specification.md
├── config/                              # 全局配置文件
│   ├── mqtt_config.json
│   ├── iot_hub_config.json
│   ├── websocket_config.json
│   └── deployment_config.yaml
├── protocols/                           # 通信协议定义
│   ├── data_packet_format.proto
│   ├── websocket_protocol.json
│   └── mqtt_topics.json
├── hardware/                            # 硬件固件项目
│   ├── stm32_firmware/
│   └── esp32_firmware/
├── qt_preprocessor/                      # Qt预处理应用
│   ├── src/
│   ├── ui/
│   ├── config/
│   └── build/
├── cloud_services/                      # 云端服务
│   ├── iot_hub_integration/
│   ├── cloud_functions/
│   ├── ai_processing/
│   ├── data_storage/
│   └── websocket_server/
├── unity_mobile/                        # Unity移动端应用
│   ├── Assets/
│   ├── ProjectSettings/
│   └── Packages/
├── deployment/                          # 部署配置
│   ├── docker/
│   ├── kubernetes/
│   └── terraform/
└── tools/                              # 开发工具
    ├── data_simulator/
    ├── testing_tools/
    └── monitoring/
```

详细项目结构解释

1. 硬件固件项目 (hardware/)

STM32固件结构

```
hardware/stm32_firmware/
├── Core/
│   ├── Inc/                                # 头文件
│   │   ├── main.h
│   │   ├── sensor_manager.h
│   │   ├── data_processor.h
│   │   ├── communication.h
│   │   └── protocol_handler.h
│   └── Src/                                # 源文件
│       ├── main.c
│       ├── sensor_manager.c               # 传感器管理
│       ├── data_processor.c              # 数据处理
│       ├── communication.c               # 通信模块
│       └── protocol_handler.c             # 协议处理
├── Drivers/                                # 驱动文件
│   ├── STM32F4xx_HAL_Driver/
│   ├── CMSIS/
│   └── Custom_Drivers/
│       ├── flex_sensor_driver.c          # 弯曲传感器驱动
│       ├── strain_gauge_driver.c         # 应变片驱动
│       ├── mpu6050_driver.c              # IMU驱动
│       └── bluetooth_driver.c            # 蓝牙驱动
├── Middlewares/                            # 中间件
│   ├── FreeRTOS/
│   ├── MQTT/
│   └── JSON_Parser/
├── Config/                                # 配置文件
│   ├── sensor_calibration.h
│   ├── communication_config.h
│   └── system_config.h
├── STM32F401RETx.ioc                       # STM32CubeMX配置
├── Makefile
└── README.md
```

ESP32固件结构

```
hardware/esp32_firmware/
├── main/
│   ├── main.cpp                           # 主程序
│   ├── sensor_controller.cpp              # 传感器控制器
│   ├── data_manager.cpp                  # 数据管理
│   ├── wifi_manager.cpp                  # WiFi管理
│   ├── mqtt_client.cpp                   # MQTT客户端
│   ├── bluetooth_handler.cpp             # 蓝牙处理
│   └── protocol_encoder.cpp               # 协议编码器
└── components/                            # 自定义组件
```

```

├── flex_sensor/
│   ├── flex_sensor.h
│   └── flex_sensor.cpp
├── strain_gauge/
│   ├── strain_gauge.h
│   └── strain_gauge.cpp
├── imu_mpu6050/
│   ├── mpu6050.h
│   └── mpu6050.cpp
├── data_packet/
│   ├── data_packet.h
│   └── data_packet.cpp
├── include/
│   ├── config.h
│   ├── sensor_definitions.h
│   └── communication_protocol.h
├── data/                                # 配置数据
│   ├── wifi_config.json
│   ├── mqtt_config.json
│   └── sensor_calibration.json
├── CMakeLists.txt
├── sdkconfig
└── README.md

```

2. Qt预处理应用 (qt_preprocessor/)

```

qt_preprocessor/
├── src/
│   ├── main.cpp
│   ├── mainwindow.cpp                # 主窗口
│   ├── controllers/
│   │   ├── data_controller.cpp        # 数据控制器
│   │   ├── communication_controller.cpp # 通信控制器
│   │   └── preprocessing_controller.cpp # 预处理控制器
│   ├── models/
│   │   ├── sensor_data_model.cpp      # 传感器数据模型
│   │   ├── gesture_model.cpp          # 手势模型
│   │   └── calibration_model.cpp       # 校准模型
│   ├── views/
│   │   ├── sensor_view.cpp            # 传感器视图
│   │   ├── calibration_view.cpp       # 校准视图
│   │   └── data_visualization.cpp      # 数据可视化
│   ├── services/
│   │   ├── bluetooth_service.cpp      # 蓝牙服务
│   │   ├── mqtt_service.cpp           # MQTT服务
│   │   ├── data_filter_service.cpp     # 数据滤波服务
│   │   └── cloud_upload_service.cpp    # 云端上传服务
│   └── utils/
│       ├── data_processor.cpp          # 数据处理工具
│       ├── kalman_filter.cpp           # 卡尔曼滤波
│       └── signal_processing.cpp       # 信号处理

```

```

├── json_serializer.cpp      # JSON序列化
├── ui/
│   ├── mainwindow.ui       # 主界面UI
│   ├── sensor_calibration.ui # 传感器校准UI
│   ├── data_monitor.ui     # 数据监控UI
│   └── settings.ui         # 设置界面UI
├── resources/
│   ├── icons/
│   ├── config/
│   │   ├── default_settings.json
│   │   └── sensor_parameters.json
│   └── qt_resources.qrc
├── config/
│   ├── app_config.ini
│   ├── communication_config.ini
│   └── preprocessing_config.ini
├── build/                  # 构建输出目录
├── CMakeLists.txt
├── qt_preprocessor.pro
└── README.md

```

3. 云端服务 (cloud_services/)

IoT Hub集成

```

cloud_services/iot_hub_integration/
├── src/
│   ├── iot_device_manager.js      # 设备管理
│   ├── message_router.js          # 消息路由
│   ├── device_shadow_handler.js   # 设备影子处理
│   └── telemetry_processor.js      # 遥测数据处理
├── config/
│   ├── iot_hub_config.json
│   ├── device_templates.json
│   └── message_rules.json
├── schemas/
│   ├── device_data_schema.json    # 设备数据模式
│   └── command_schema.json        # 命令模式
├── package.json
└── README.md

```

云函数

```

cloud_services/cloud_functions/
├── gesture_analysis/
│   ├── index.js                  # 手势分析函数入口
│   └── src/
│       ├── data_preprocessor.js  # 数据预处理
│       ├── feature_extractor.js  # 特征提取
│       └── gesture_classifier.js # 手势分类

```

```

├── result_formatter.js      # 结果格式化
├── models/
│   ├── ml_model.json      # 机器学习模型
│   └── gesture_dictionary.json # 手势字典
├── config/
│   └── function_config.json
├── package.json
├── data_storage/
│   ├── index.js           # 数据存储函数
│   ├── src/
│   │   ├── database_writer.js # 数据库写入
│   │   ├── data_validator.js  # 数据验证
│   │   └── backup_manager.js  # 备份管理
│   └── package.json
├── realtime_notification/
│   ├── index.js           # 实时通知函数
│   ├── src/
│   │   ├── websocket_manager.js # WebSocket管理
│   │   ├── push_notification.js  # 推送通知
│   │   └── message_formatter.js  # 消息格式化
│   └── package.json
├── deployment/
│   ├── serverless.yml     # Serverless配置
│   └── function_config.yaml # 函数配置

```

AI处理服务

```

cloud_services/ai_processing/
├── gesture_recognition/
│   ├── model_training/
│   │   ├── train.py        # 模型训练脚本
│   │   ├── data_loader.py  # 数据加载器
│   │   ├── model_architecture.py # 模型架构
│   │   └── evaluation.py    # 模型评估
│   ├── inference/
│   │   ├── gesture_predictor.py # 手势预测器
│   │   ├── model_loader.py      # 模型加载器
│   │   └── postprocessor.py     # 后处理
│   ├── models/
│   │   ├── trained_models/     # 训练好的模型
│   │   └── model_configs/      # 模型配置
│   └── data/
│       ├── training_data/      # 训练数据
│       └── validation_data/    # 验证数据
├── feature_engineering/
│   ├── feature_extractor.py    # 特征提取器
│   ├── signal_processor.py     # 信号处理
│   ├── data_normalizer.py     # 数据归一化
│   └── feature_selector.py     # 特征选择
├── api_service/
│   └── app.py                  # Flask应用

```

```

├── routes/
│   ├── prediction_api.py      # 预测API
│   ├── training_api.py       # 训练API
│   └── model_management_api.py # 模型管理API
├── utils/
│   ├── response_formatter.py  # 响应格式化
│   └── error_handler.py      # 错误处理
├── requirements.txt
├── Dockerfile
└── README.md

```

WebSocket服务器

```

cloud_services/websocket_server/
├── src/
│   ├── server.js              # WebSocket服务器主文件
│   ├── connection_manager.js  # 连接管理
│   ├── message_handler.js     # 消息处理
│   ├── room_manager.js        # 房间管理
│   └── authentication.js      # 认证模块
├── middleware/
│   ├── rate_limiter.js        # 限流中间件
│   ├── logger.js              # 日志中间件
│   └── error_handler.js       # 错误处理
├── config/
│   ├── server_config.json     # 服务器配置
│   └── websocket_config.json  # WebSocket配置
├── tests/
│   ├── connection_test.js     # 连接测试
│   └── message_test.js        # 消息测试
├── package.json
├── Dockerfile
└── README.md

```

4. Unity移动端应用 (unity_mobile/)

```

unity_mobile/
├── Assets/
│   ├── Scripts/
│   │   ├── Core/
│   │   │   ├── GameManager.cs    # 游戏管理器
│   │   │   ├── DataManager.cs    # 数据管理器
│   │   │   └── SceneManager.cs    # 场景管理器
│   │   ├── Networking/
│   │   │   ├── WebSocketClient.cs # WebSocket客户端
│   │   │   ├── APIClient.cs       # API客户端
│   │   │   └── DataSynchronizer.cs # 数据同步器
│   │   └── UI/
│   │       ├── MainMenuUI.cs      # 主菜单UI
│   │       └── GestureDisplayUI.cs # 手势显示UI

```

```

├── CalibrationUI.cs    # 校准UI
├── SettingsUI.cs      # 设置UI
├── Gesture/
│   ├── GestureRenderer.cs # 手势渲染器
│   ├── HandModel.cs      # 手部模型
│   ├── FingerController.cs # 手指控制器
│   └── GestureAnimator.cs # 手势动画器
├── Data/
│   ├── SensorData.cs     # 传感器数据结构
│   ├── GestureData.cs    # 手势数据结构
│   └── ConfigData.cs     # 配置数据结构
├── Utils/
│   ├── JsonUtility.cs    # JSON工具
│   ├── MathUtility.cs    # 数学工具
│   └── DebugUtility.cs   # 调试工具
├── Prefabs/
│   ├── UI/               # UI预制体
│   ├── Hand/             # 手部预制体
│   └── Effects/          # 特效预制体
├── Materials/
│   ├── HandMaterials/    # 手部材质
│   └── UIMaterials/      # UI材质
├── Models/
│   ├── HandModels/       # 手部模型
│   └── UIModels/         # UI模型
├── Textures/
│   ├── HandTextures/     # 手部纹理
│   └── UITextures/       # UI纹理
├── Animations/
│   ├── HandAnimations/   # 手部动画
│   └── UIAnimations/     # UI动画
├── Scenes/
│   ├── MainScene.unity   # 主场景
│   ├── CalibrationScene.unity # 校准场景
│   └── TestingScene.unity # 测试场景
├── StreamingAssets/
│   ├── config.json       # 配置文件
│   └── gesture_templates.json # 手势模板
├── ProjectSettings/      # 项目设置
├── Packages/             # 包管理
│   └── manifest.json
└── README.md

```

5. 通信协议定义 (protocols/)

```

protocols/
├── data_packet_format.proto      # Protocol Buffers定义
├── websocket_protocol.json      # WebSocket协议规范
├── mqtt_topics.json            # MQTT主题定义
├── api_endpoints.json          # API端点定义
├── message_formats/
│   ├── sensor_data_format.json  # 传感器数据格式
│   ├── gesture_result_format.json # 手势结果格式
│   ├── command_format.json      # 命令格式
│   └── error_format.json        # 错误格式

```

6. 部署配置 (deployment/)

```

deployment/
├── docker/
│   ├── docker-compose.yml      # Docker Compose配置
│   ├── Dockerfile.websocket    # WebSocket服务Dockerfile
│   ├── Dockerfile.ai_service   # AI服务Dockerfile
│   └── Dockerfile.api_gateway   # API网关Dockerfile
├── kubernetes/
│   ├── namespace.yaml          # 命名空间
│   ├── configmap.yaml          # 配置映射
│   ├── secrets.yaml            # 密钥
│   ├── deployments/
│   │   ├── websocket-deployment.yaml
│   │   ├── ai-service-deployment.yaml
│   │   └── api-gateway-deployment.yaml
│   └── services/
│       ├── websocket-service.yaml
│       ├── ai-service-service.yaml
│       └── api-gateway-service.yaml
├── terraform/
│   ├── main.tf                 # 主配置
│   ├── variables.tf            # 变量定义
│   ├── outputs.tf              # 输出定义
│   ├── modules/
│   │   ├── tencent_cloud/
│   │   │   ├── iot_hub/
│   │   │   └── database/
│   └── environments/
│       ├── dev.tfvars          # 开发环境变量
│       ├── staging.tfvars      # 测试环境变量
│       └── prod.tfvars         # 生产环境变量
├── scripts/
│   ├── deploy.sh               # 部署脚本
│   ├── setup_env.sh            # 环境设置脚本
│   └── backup.sh                # 备份脚本

```

核心通信协议实现

WebSocket协议示例

```
// protocols/websocket_protocol.json
{
  "protocol_version": "1.0",
  "message_types": {
    "sensor_data": {
      "type": "sensor_data",
      "timestamp": "number",
      "device_id": "string",
      "data": {
        "flex_sensors": "array[5]",
        "strain_gauge": "number",
        "imu": {
          "acceleration": "array[3]",
          "gyroscope": "array[3]",
          "magnetometer": "array[3]"
        }
      }
    },
    "gesture_result": {
      "type": "gesture_result",
      "timestamp": "number",
      "gesture_id": "string",
      "gesture_name": "string",
      "confidence": "number",
      "translation": "string"
    },
    "command": {
      "type": "command",
      "timestamp": "number",
      "command": "string",
      "parameters": "object"
    },
    "error": {
      "type": "error",
      "timestamp": "number",
      "error_code": "string",
      "error_message": "string"
    }
  }
}
```

MQTT主题结构

```
// protocols/mqtt_topics.json
{
  "topics": {
    "sensor_data": {
      "publish": "dataglove/{device_id}/sensor/data",
    }
  }
}
```

```
    "subscribe": "dataglove/{device_id}/sensor/command"
  },
  "gesture_result": {
    "publish": "dataglove/{device_id}/gesture/result",
    "subscribe": "dataglove/{device_id}/gesture/feedback"
  },
  "device_status": {
    "publish": "dataglove/{device_id}/status",
    "subscribe": "dataglove/{device_id}/control"
  },
  "calibration": {
    "publish": "dataglove/{device_id}/calibration/data",
    "subscribe": "dataglove/{device_id}/calibration/command"
  }
},
"qos_levels": {
  "sensor_data": 0,
  "gesture_result": 1,
  "device_status": 1,
  "calibration": 2
}
}
```

项目特点说明

1. 模块化设计

- 每个子系统独立开发和部署
- 清晰的接口定义和协议规范
- 支持独立测试和调试

2. 实时性保障

- WebSocket确保低延迟通信
- MQTT提供可靠的设备通信
- 云函数支持毫秒级响应

3. 可扩展性

- 支持多设备接入
- AI模型可独立更新
- 支持水平扩展

4. 数据处理流程

传感器数据 → 硬件预处理 → Qt滤波校准 → 云端AI分析 → Unity渲染显示

5. 容错机制

- 多层次的错误处理
- 数据备份和恢复
- 离线模式支持

这个项目结构提供了完整的开发框架，支持从原型开发到生产部署的全生命周期管理。