文档编号：

|  |  |  |  |
| --- | --- | --- | --- |
|  | 系统类别 |  | 共 页  （含封页） |
| 型号 |  |
| 名称 |  |

**起重机项目设计方案**（文档版本号：V1.1）

|  |  |  |  |
| --- | --- | --- | --- |
| **拟 制：** |  | **日 期：** | **2023/03/10** |
| **审 核：** |  | **日 期：** |  |
| **会 签：** |  | **日 期：** |  |
| **批 准：** |  | **日 期：** |  |

**陕西伟景机器人**

版权所有 不得复制

修订记录

|  |  |  |  |
| --- | --- | --- | --- |
| 日 期 | 修订版本 | 描 述 | 作 者 |
| 2023/03/10 | V1.0 | 初版 | 谭创 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

目录

[1 系统环境 4](#_Toc129354562)

[1.1 运行环境 4](#_Toc129354563)

[1.2 开发环境 4](#_Toc129354564)

[2 软件设计 4](#_Toc129354565)

[2.1 整体架构 4](#_Toc129354566)

[2.2 上位机设计 4](#_Toc129354567)

[2.2.1 流程图 5](#_Toc129354568)

[2.2.2 接口定义 5](#_Toc129354569)

[2.2.3 源码分层 11](#_Toc129354570)

[2.3 中间件 14](#_Toc129354571)

[2.3.1 流程图 14](#_Toc129354572)

[2.3.2 环境编译 14](#_Toc129354573)

[2.3.3 接口设计 18](#_Toc129354574)

[2.4 数据库 24](#_Toc129354575)

[2.4.1 接口设计 24](#_Toc129354576)

[2.4.2 接口说明 25](#_Toc129354577)

[2.4.3 测试数据展示页 25](#_Toc129354578)

# 系统环境

## 运行环境

Windows10 x64 系统 （最好配备独立显卡）

## 开发环境

Qt5.14.2+msvc2017 编译

boost-1\_74

flatbuffers-1.12.0

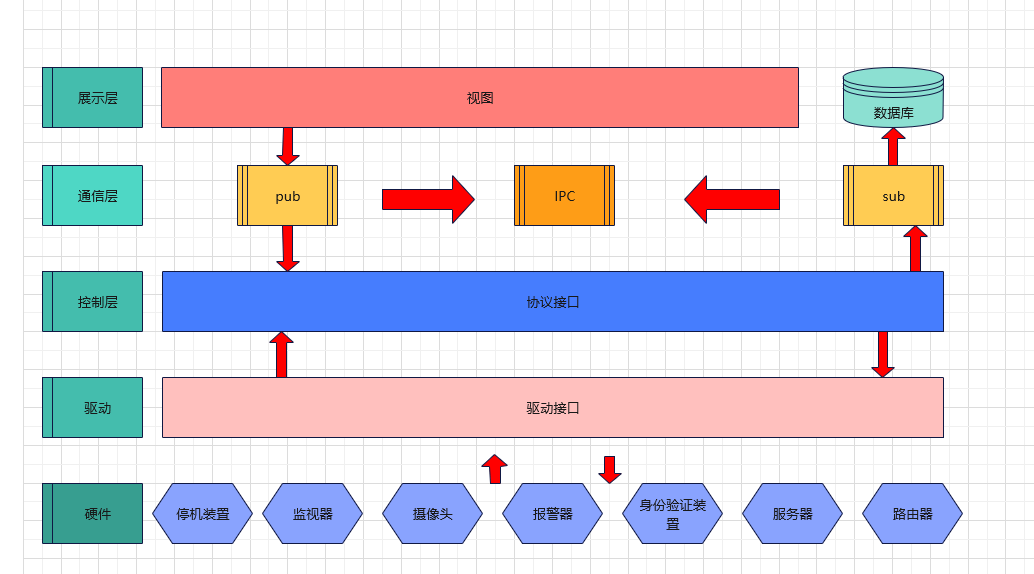
ZeroMq-4-3-2

mysql-5.7.27-winx64

sqlit-Win64

# 整体方案

## 方案图



# 软件设计

无

## 整体架构

针对本产品，软件总体分为三大模块，上位机展示模块，中间件数据交互模块，数据库记录模块

## 上位机设计

### 流程图

图示

描述已自动生成

### 接口定义

线程模块基础接口：

图形用户界面, 文本, 应用程序

描述已自动生成

MVC基础接口：

1.控制器

图形用户界面, 文本, 应用程序

中度可信度描述已自动生成

2.模型

图形用户界面, 文本

描述已自动生成

3.视图

文本

描述已自动生成

基础界面：

文本

描述已自动生成

图形用户界面, 文本, 应用程序

描述已自动生成

崩溃捕捉：



文本

描述已自动生成

### 源码分层

图片包含 图形用户界面

描述已自动生成

以下是主要模块，其余模块为外部开源库，以及资源模块

图片包含 日程表

描述已自动生成

1. dataInterFace

product产品类（数据吞吐模块）

1. thread

IPC监管线程，如果服务停止，上报状态

数据库监管线程，数据量超过极限值自动清空历史数据

工作线程，检查消息上报是否正常

图形用户界面, 文本, 应用程序, 聊天或短信

描述已自动生成

1. jt808数据编解码类
2. OpenGLWidegt 摄像头画面显示基类（当前方法是使用rstp协议，拉流解码yuv opengwidget渲染）
3. StreamThread 摄像头解码线程
4. TGFrame 资源文件加载界面

图形用户界面, 文本

描述已自动生成

图形用户界面, 应用程序

描述已自动生成

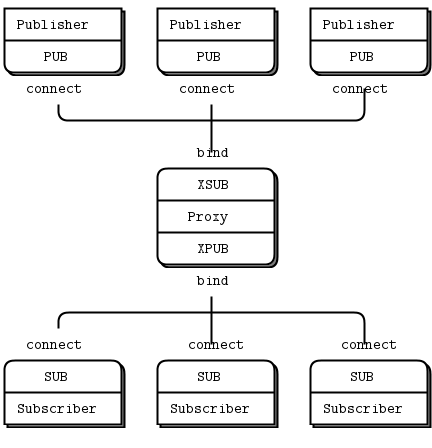
表格

中度可信度描述已自动生成

Ui模块，主要是mvc分层界面

## 中间件

### 流程图



### 环境编译

**1环境**

* 1. 安装包：





1.2 安装步骤：

文本

描述已自动生成

如图表示cmake安装成功

Flatc 编译生成：

如图解压flatbuffer源码包

图片包含 文本

描述已自动生成

新建build目录进入



执行 

在build目录中生成sln的工程 如图：

表格

描述已自动生成

使用vs2017打开项目并编译生成Release

图形用户界面, 文本, 应用程序

描述已自动生成

**1.3 fbs编译**

Flatc 的使用：

编辑fbs文件 例：test.fbs

文本, 信件

描述已自动生成



如图编译fbs

flatc.exe --cpp test.fbs --gen-object-api --gen-compare （生成带T 文件）

注：这里必须加参数--gen-object-api --gen-compare 为了生成带T的结构体，方便序列化与反序列化

如图生成



### 接口设计

# ZMQ Wrapper Design

## 遇到的问题 & 解决方案

- 因消息通道过多导致zmq通道名称/端口号/接收线程资源等被大量占用

在实际使用中, 我们对每条数据都创建独立的消息通道, 导致占用了大量端口号/线程等资源

![avatar](image/message\_before.drawio.png)

优化后我们将采用XPUB-XSUB模型(如下)

![avatar](image/message\_after.drawio.png)

- 消息通道大多是一对一链接, 存在大量数据转发的情况

## ZMQ消息模型

  1、与pub-sub模式雷同，但有些区别

      2、ZMQ\_XPUB不仅支持send方法，还支持recv方法，但是只能接收subscribe消息和unsubscribe消息，接收不到其他消息

     3、ZMQ\_XSUB不仅支持recv方法，还支持send方法，但是只能发送subscribe消息和unsubscribe消息，不能发送其他消息

     4、数据流向是单向的——从XPUB流向XSUB；

     5、订阅/取消 订阅是单向的——从XSUB到XPUB；

     6、一个ZMQ\_XSUB可以订阅多个ZMQ\_XPUB的数据。

### 消息池定义(node.cpp)

- inproc\_pool: 线程消息池, 用于线程间通信

inproc\_sub\_channel: inproc://sub\_channel

inproc\_pub\_channel: inproc://pub\_channel

- tcp\_pool: 网络消息池, 用于跨节点通信

tcp\_sub\_channel: tcp://0.0.0.0:7000

tcp\_pub\_channel: tcp://0.0.0.0:7001

### 接口定义

- 回调函数定义(zmq\_wrapper.h)

namespace datax {

using SubscribeFunc = std::function<void(const char\*, const void\*, uint32\_t)>;

}

- 订阅/取消指定主题并指定回调函数(zmq\_wrapper.h)

void Subscribe(std::string topic, SubscribeFunc func);

void UnSubscribe(std::string topic);

- 订阅/取消全部主题并指定回调函数(zmq\_wrapper.h)

void SubscribeAll(SubscribeFunc func);

void UnSubscribeAll(void);

- 发送指定主题的消息

size\_t Post(std::string topic, const void \*payload, size\_t payload\_len);

- 订阅/取消指定主题并指定回调函数(将基于Message类型进行解析)(zmq\_wrapper.h)

void SubscribeWithMessage(std::string topic, SubscribeFunc func);

void UnSubscribeWithMessage(std::string topic);

- 发送指定主题消息(将基于Message类型进行封包)(zmq\_wrapper.h)

size\_t PostWithMessage(std::string topic, const void \*payload, size\_t payload\_len);

- 提供各消息池指针(inproc/tcp)

std::shared\_ptr<ZmqWrapper> GetIpcPool(void);

std::shared\_ptr<ZmqWrapper> GetInprocPool(void);

std::shared\_ptr<ZmqWrapper> GetTcpPool(void);

### 示例代码

- fbs定义

table Repeat{

id:int (id:0);

}

table MsgTest{

result:ubyte (id:0);

time:uint64(id:1);

array:[Repeat](id:2);

}

- send

MsgTestT result;

result.result = 1;

// 打包 MsgTest

flatbuffers::FlatBufferBuilder fbb;

fbb.Finish(CreateMsgTest(fbb, &result));

MessageT msg;

msg.timestamp = 12345;

msg.node\_name = "sender";

msg.data.assign(fbb.GetBufferPointer(), fbb.GetBufferPointer() + fbb.GetSize());

msg.size = fbb.GetSize();

// 打包 Message

flatbuffers::FlatBufferBuilder builder;

builder.Finish(CreateMessage(builder, &msg));

datax::Node sender;

// Post 接口对发送内容不做处理

sender.Post(TOPIC\_IPC\_MESSAGE\_TEST\_TOPIC, builder.GetBufferPointer(), builder.GetSize());

// PostWithMessage 接口会将发送内容以 Message 打包后发出

// 此处与上方 Post 接口等价

sender.PostWithMessage(TOPIC\_IPC\_MESSAGE\_TEST\_TOPIC, fbb.GetBufferPointer(), fbb.GetSize());

- recv

Subscribe 接口:

datax::Node recver;

void CallBackFun(const char\* topic, const void\*payload, uint32\_t len)

{

auto msg = GetMessage(payload);

auto content = flatbuffers::GetRoot<MsgTest>(msg->data()->Data());

printf("content result: %d\n", content->result());

}

recver.Subscribe(TOPIC\_IPC\_MESSAGE\_TEST\_TOPIC, CallBackFun);

SubscribeWithMessage 接口:

void CallBackFun2(const char\* topic, const void\*payload, uint32\_t len)

{

auto content = flatbuffers::GetRoot<MsgTest>(payload);

printf("content result: %d\n", content->result());

}

// SubscribeWithMessage 接口接收数据是会按 Message 格式解析

// 此处与上方 Subscribe 接口等价

recver.SubscribeWithMessage(TOPIC\_IPC\_MESSAGE\_TEST\_TOPIC, CallBackFun2);

SubscribeTemplate 接口:

std::function<void(const MsgTest\*)> f = [](const MsgTest\* msg) {

if(msg == nullptr)

{

printf("recv null!\n");

}

else

{

printf("result: %d\n", msg->result());

}

};

// SubscribeTemplate 接口支持模板函数回调

// 回调函数为std::function类型, 参数需定义为flatbuffer结构指针

// 发送端需二次封装MessageT结构, 或可直接使用PostWithMessage发送接口

recver.SubscribeTemplate(TOPIC\_IPC\_MESSAGE\_TEST\_TOPIC, f);

### 注意

- 回调函数中禁止耗时操作, 只做事件同步

- flatbuffer -> 维护类型列表 -> 维护topic:type映射表

## 数据库

### 接口设计

文本, 应用程序

描述已自动生成

图片包含 文本

描述已自动生成

### 接口说明

数据库接口设计无非是增删改查，目前所有常用的接口都已经封装出来，后续会把接口全部导出作为独立的sdk调用，同时将driver驱动库一起打包，然后平行移植调用

### 测试数据展示页

图形用户界面, 应用程序, 表格

描述已自动生成

当前为数据库写入的测试数据，存储回放展示功能