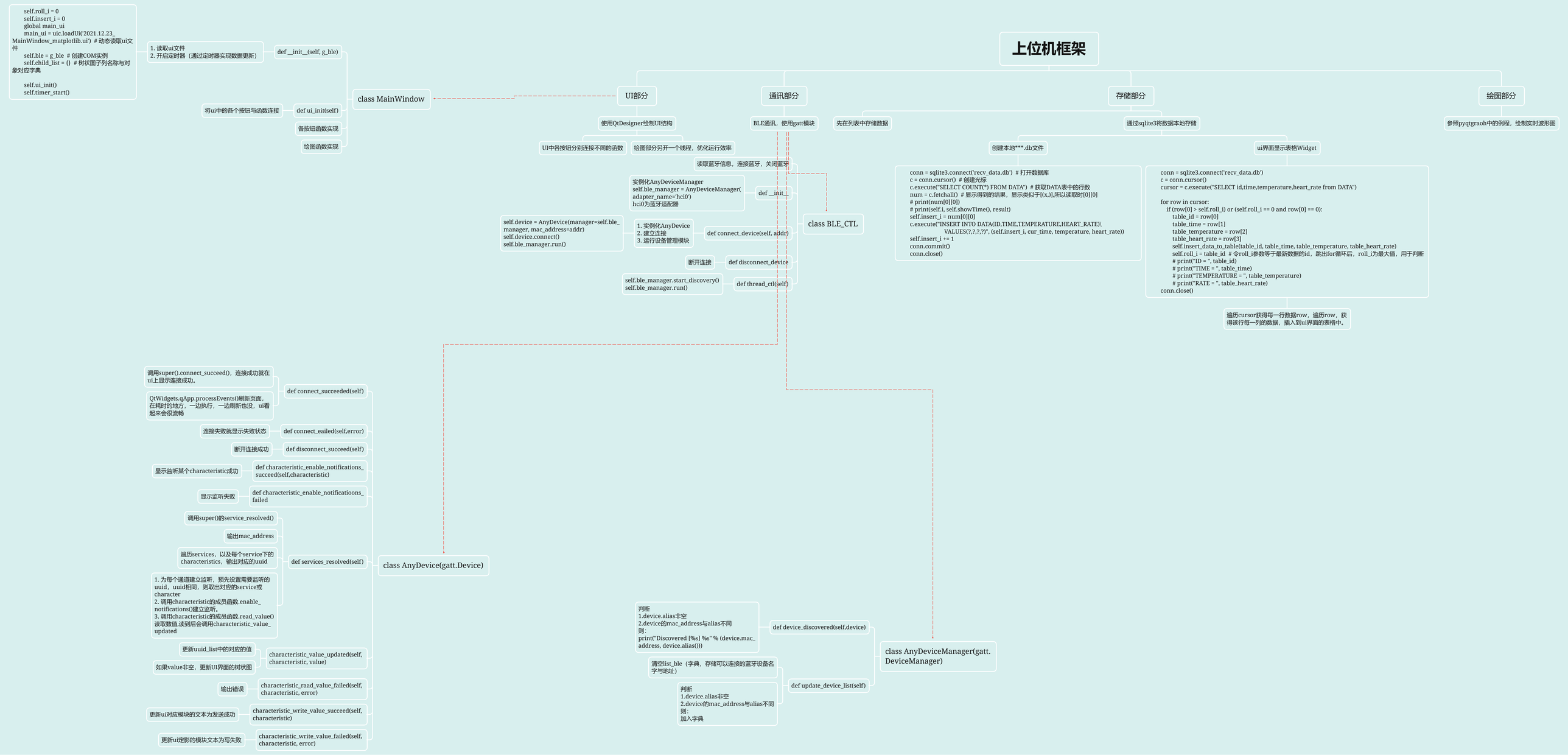
上位机框架
UI部分
通讯部分
存储部分
绘图部分
class MainWindow
def __init__(self, g_ble)
1. 读取ui文件
2. 开启定时器（通过定时器实现数据更新）
def ui_init(self)
将ui中的各个按钮与函数连接
各按钮函数实现
绘图函数实现
self.roll_i = 0
self.insert_i = 0
global main_ui
main_ui = uic.loadUi('2021.12.23_MainWindow_matplotlib.ui') # 动态读取ui文件
self.ble = g_ble # 创建COM实例
self.child_list = {} # 树状图子列名称与对象对应字典
self.ui_init()
self.timer_start()
使用QtDesigner绘制UI结构
UI中各按钮分别连接不同的函数
绘图部分另开一个线程，优化运行效率
BLE通讯，使用gatt模块
读取蓝牙信息，连接蓝牙，关闭蓝牙
class BLE_CTL
def __init__
实例化AnyDeviceManager
self.ble_manager = AnyDeviceManager(adapter_name='hci0')
hci0为蓝牙适配器
def connect_device(self, addr)
1. 实例化AnyDevice
2. 建立连接
3. 运行设备管理模块
self.device = AnyDevice(manager=self.ble_manager, mac_address=addr)
self.device.connect()
self.ble_manager.run()
def disconnect_device
断开连接
def thread_ctl(self)
self.ble_manager.start_discovery()
self.ble_manager.run()
先在列表中存储数据
通过sqlite3将数据本地存储
创建本地***.db文件
conn = sqlite3.connect('recv_data.db') # 打开数据库
c = conn.cursor() # 创建光标
c.execute("SELECT COUNT(*) FROM DATA") # 获取DATA表中的行数
num = c.fetchall() # 显示得到的结果，显示类似于[(x,)],所以读取时[0][0]
# print(num[0][0])
# print(self.i, self.showTime(), result)
self.insert_i = num[0][0]
c.execute("INSERT INTO DATA(ID,TIME,TEMPERATURE,HEART_RATE)\
VALUES(?,?,?,?)", (self.insert_i, cur_time, temperature, heart_rate))
self.insert_i += 1
conn.commit()
conn.close()
ui界面显示表格Widget
conn = sqlite3.connect('recv_data.db')
c = conn.cursor()
cursor = c.execute("SELECT id,time,temperature,heart_rate from DATA")
for row in cursor:
if (row[0] > self.roll_i) or (self.roll_i == 0 and row[0] == 0):
table_id = row[0]
table_time = row[1]
table_temperature = row[2]
table_heart_rate = row[3]
self.insert_data_to_table(table_id, table_time, table_temperature, table_heart_rate)
self.roll_i = table_id # 令roll_i参数等于最新数据的id，跳出for循环后，roll_i为最大值，用于判断
# print("ID = ", table_id)
# print("TIME = ", table_time)
# print("TEMPERATURE = ", table_temperature)
# print("RATE = ", table_heart_rate)
conn.close()
遍历cursor获得每一行数据row，遍历row，获得该行每一列的数据，插入到ui界面的表格中。
参照pyqtgraoh中的例程，绘制实时波形图
class AnyDevice(gatt.Device)
def connect_succeeded(self)
调用super().connect_succeed()，连接成功就在ui上显示连接成功。
QtWidgets.qApp.processEvents()刷新页面，在耗时的地方，一边执行，一边刷新也没，ui看起来会很流畅
def connect_eailed(self,error)
连接失败就显示失败状态
def disconnect_succeed(self)
断开连接成功
def characteristic_enable_notifications_succeed(self,characteristic)
显示监听某个characteristic成功
def characteristic_enable_notificatioons_failed
显示监听失败
def services_resolved(self)
调用super()的service_resolved()
输出mac_address
遍历services，以及每个service下的characteristics，输出对应的uuid
1. 为每个通道建立监听，预先设置需要监听的uuid，uuid相同，则取出对应的service或character
2. 调用characteristic的成员函数.enable_notifications()建立监听。
3. 调用characteristic的成员函数.read_value()读取数值.读到后会调用characteristic_value_updated
characteristic_value_updated(self, characteristic, value)
更新uuid_list中的对应的值
如果value非空，更新UI界面的树状图
characteristic_raad_value_failed(self, characteristic, error)
输出错误
characteristic_write_value_succeed(self, characteristic)
更新ui对应模块的文本为发送成功
characteristic_write_value_failed(self, characteristic, error)
更新ui定影的模块文本为写失败
class AnyDeviceManager(gatt.DeviceManager)
def device_discovered(self,device)
判断
1.device.alias非空
2.device的mac_address与alias不同
则：
print("Discovered [%s] %s" % (device.mac_address, device.alias()))
def update_device_list(self)
清空list_ble（字典，存储可以连接的蓝牙设备名字与地址）
判断
1.device.alias非空
2.device的mac_address与alias不同
则：
加入字典