

《物联网综合实验》

设计报告

（ 2024 / 2025 学年 第2学期）

题 目： **智慧停车场仿真系统设计与实现**

**学号姓名**

**指导教师**

**指导单位 物联网学院**

**日 期 2025年6月18日**

|  |  |  |
| --- | --- | --- |
| **课程评分细则** | **评分项** | **成绩** |
| **平时成绩（10分）：**  **从遵守机房规章制度、上机时的表现、学习态度等方面考察。** |  |
| **基本功能演示(20分)：**  **验证性实验功能实现情况。** |  |
| **扩展功能实现情况(30分)：**  **扩展功能是否实现，问题回答准确度，功能实用性及创新性等** |  |
| **设计报告（30分）：**  **从报告书写认真程度、内容详实程度、文字表达熟练程度、源代码分析的翔实程度、内容是否全面等方面考察。** |  |
| **总计（100分）：** |  |
| **简短评语** | **教师签名：**  **2025年 7月 2 日** | |
| **评分等级** |  | |
| **备注** | **评分等级有五种：优秀、良好、中等、及格、不及格** | |

**一、需求分析**

本项目实现了一个基于客户端-服务器架构的停车场模拟系统，分为服务器端和客户端两部分，分别模拟停车场和汽车的行为。

我使用的是c++语言开发，计划使用c++11新标准的智能指针来管理对象的生命周期，尽可能避免内存泄漏。多次使用lambda表达式，使代码更加内聚，提高程序可读性。由于实验周时间紧张，只有两周去完成项目和报告，而且这个项目没有实际工程化意义和需求，所以我放弃了使用多线程并发的设计。

因为多线程涉及到加锁，而项目中很多数据结构被频繁的调用，会导致并发量大大降低。需要尽可能减少锁的使用。这就需要CAS操作，使用原子操作代替普通的智能指针，使用无锁队列代替朴素的加锁操作。同时，锁的粒度很大，这就需要哈希分片管理存储信息的数据结构。由于哈希分片导致的负载不均衡又需要更多的技术解决。由于项目用到了简单的日志方便debug，多线程环境下的日志系统需要用双缓冲区优化，这又是一个复杂的项目。多线程带来的问题可以参考github上陈硕的muduo网络库的实现，我两周时间实在难以完成。所以这个项目用的是事件驱动的单线程异步通信，放弃了支持高QPS的多线程设计。

下面是具体的需求分析，本项目系统结构上应该分为服务端和客户端。

服务器端负责模拟停车场的运行，管理停车位的分配与释放，并通过图形化界面实时显示停车场状态。客户端中每个客户端模拟一辆汽车，向服务器端发起停车请求，并根据服务器端的反馈决定是否停车或离开。

从功能需求角度，服务器端有停车位管理，即服务器端维护n个停车位（n为可配置参数），记录每个车位的占用情况。有请求处理，接收来自客户端（汽车）的停车请求，根据当前车位状态进行处理。有车位分配策略：汽车到达时，若有空闲车位，则分配车位并允许停车。若无空闲车位，则通知客户端等待10秒后再次尝试。若10秒后仍无空闲车位，则通知客户端离开。实时图形化显示，通过图形化界面（如Qt界面）实时显示每个车位的状态（空闲/已占用），并动态更新。

客户端包含汽车模拟，即每个客户端代表一辆汽车，随机时间到达停车场，随机时间请求离开。停车请求即到达停车场后，向服务器端发送停车请求。还包含等待与离开机制，即若收到服务器端允许停车的反馈，则进入停车状态。若收到等待通知，则等待10秒后再次请求。若10秒后仍无空位，则放弃停车，离开停车场。离开请求：停车一段随机时间后，向服务器端发送离开请求，释放车位。

从性能与并发角度，系统支持多个客户端同时与服务器端通信，服务器端能够正确处理并发的停车与离开请求，保证车位分配的正确性和一致性。

从用户界面角度，服务器端提供直观的图形化界面，实时反映停车场的车位变化，便于观察和管理。

同时系统应具备良好的可扩展性，便于后续增加更多功能。代码应该结构清晰，便于维护和升级。实际的主界面ui设计如图1所示：

图1 智慧停车场仿真系统的ui设计

**二、系统架构**

这个项目是一个基于Qt框架的图形化停车场管理系统，采用MVC架构模式，包含以下核心组件：Qt 5+ (Core, GUI, Widgets, Network)，C++11 智能指针，TCP网络通信，定时器机制。

表示层（UI层）有MainWindow类，负责主窗口界面，继承自QMainWindow。UI控件负责停车位可视化显示、车牌输入框、操作按钮等。事件处理负责按钮点击事件、网络消息处理等

业务逻辑层有车辆进出管理，即处理车辆进入和离开停车场的核心逻辑。队列管理当停车场满时，车辆进入等待队列，同时进行费用计算，根据停车时长计算费用，这也不离开随机生成车牌号的功能。

数据层有Table数据结构管理停车场中的车辆信息，有Queue数据结构管理等待入场的车辆队列，也有智能指针管理使用std::unique\_ptr管理内存。

网络通信层有QTcpServer监听TCP连接，支持远程车辆管理，和QTcpSocket：处理客户端连接和消息。

定时器系统有队列超时检测定期检查等待队列中的车辆，和自动离场定时器，即每辆车都有独立的自动离场定时器。架构如图2所示：

图示

AI 生成的内容可能不正确。

图2 智慧停车场系统架构图

核心功能上，停车场管理可以动态生成停车位，可视化显示停车状态。队列管理在当停车场满时，车辆自动进入等待队列（最大容量5辆）。费用计算基于停车时长的分级收费系统，网络通信支持TCP远程控制，可接收IN:/OUT:指令。定时器系统支持队列超时检测和车辆自动离场，车牌生成函数随机生成中国标准车牌号。

项目在系统设计方面的优势十分明显，如智能指针管理，使用std::unique\_ptr自动管理内存，防止内存泄漏。循环队列可以实现高效的等待队列管理，事件驱动架构采用基于Qt信号槽机制的异步事件处理，同时采取模块化设计，即数据结构与业务逻辑分离，代码可维护性强，网络扩展上支持多客户端连接，可扩展为分布式系统息。具体的系统工作流程如图3所示：

图片包含 图表

AI 生成的内容可能不正确。

图3 智慧停车场时序图

**三、系统概要设计**

系统功能主要由mainwindow.cpp、queuefunction.cpp和tablefunction.cpp三个文件实现，接下来一次分析。

主控制模块 (mainwindow.cpp)是系统的核心控制器，负责协调各个子模块，处理用户界面交互和业务逻辑。

其关系图如图4所示。

图示

AI 生成的内容可能不正确。

图4 mainwindow核心类关系图

queuefunction.cpp是一个循环队列的实现，专门用于停车场车辆等待区的管理系统。

系统架构方面，采用循环队列结构，使用数组实现，用于管理停车场等待区的车辆排队，存储车牌号和进入时间。

队列状态检查功能检查队列是否为空或已满，队列初始化功能可以创建和清空队列。

有新车辆加入等待队列和处理队首车辆离开功能。也有根据车牌号查找特定车辆的功能。技术特性上使用循环队列避免空间浪费，使用 std::make\_unique 智能指针进行内存管理使用 QString 处理字符串数据。

具体如图5所示：

图示

AI 生成的内容可能不正确。

图5 queuefunction核心类关系图

tablefunction设计更简单一点，数据结构停车场表（table）包含车辆数组、当前车辆数量和停车场容量。状态管理提供停车场空满状态检查和初始化功能，车辆管理支持车辆进入、离开和查找操作。有基于停车时长计算费用的计费系统和随机分配可用停车位的算法。

核心功能如状态检查模块，即判断停车场是否为空或已满；车辆进出模块，即处理车辆的进入和离开操作；查询模块，即根据车牌号查找车辆；计费模块，即根据停车时长计算费用；车位管理模块，即动态分配和管理停车位示。

由于这一块功能实现比较零碎，所以使用流程图展现。

Tablefunction流程图如图6所示：

图示

AI 生成的内容可能不正确。

图6 tablefunction架构图

**四、系统详细设计**

服务端和客户端的通信是物联网综合实验最关键的模块，服务端具体实现了TCP服务器的客户端连接处理机制。

当有新的客户端连接到服务器时，程序会为每个连接建立一个QTcpSocket对象并添加到clients列表中。

对于每个新连接的客户端，程序设置了消息读取事件处理器，当客户端发送数据时会循环读取所有可用的数据行，然后根据消息格式进行解析。

如果消息以"IN:"开头，表示车辆要进入停车场，程序会提取车牌号并自动调用进车函数；如果消息以"OUT:"开头，表示车辆要离开停车场，同样提取车牌号并调用出车函数。

当客户端断开连接时，程序会自动处理断开事件，将该客户端从clients列表中移除并释放相关资源。

整个连接建立过程还包含了日志记录功能，会将当前连接的客户端数量记录到调试日志中，便于监控和调试服务器的连接状态。

这种设计允许多个客户端同时连接到停车场管理系统，实现远程车辆进出场的自动化控制。

具体服务端代码如图7所示：

文本

AI 生成的内容可能不正确。

图7 服务端通信代码实现图

客户端程序通过TCP协议与服务端进行通信，连接到本地127.0.0.1的12345端口。在构造函数中建立了三个信号槽连接，其中connect(&socket, &QTcpSocket::connected, this, &Client::onConnected)用于监听连接成功事件，connect(&socket, &QTcpSocket::readyRead, this, &Client::onReadyRead)用于监听服务端发送的数据。这里只给出简要的实现，如图8所示：

文本

AI 生成的内容可能不正确。

图8 客户端通信代码实现图

生成车牌号函数被客户端和服务端同时调用，这个函数 `generateLicensePlate` 依次完成以下操作：首先构造包含多个省份简称的 `QStringList`，随后通过 `QRandomGenerator::global()->bounded(provinces.size())` 随机选取其中一个省份代码；接着利用 `QChar('A' + ... )` 生成介于 `A` 到 `J` 的随机大写字母；然后创建字符串 `numberPart` 并追加一个 0–9 范围的随机数字；最后把省份代码、大写字母以及数字依次拼接，形成完整的车牌号并返回，如图9所示：

图片包含 文本

AI 生成的内容可能不正确。

图9 随机生成车牌号代码实现图

函数 `generateParkingSlots` 首先获取停车区域的当前布局，如果布局存在则循环删除其中的所有控件和布局项，然后删除整个旧布局以避免重复添加；接着创建新的 `QGridLayout` 布局对象，设定每行5个车位并计算所需列数，同时调用 `createTable(numSlots)` 初始化停车场数据结构；在循环中为每个车位创建 `QLabel` 标签，设置标签文本为"parkinglot"加序号，加载停车位图片并设置居中对齐，固定标签尺寸为50x50像素，然后通过 `i / cols` 和 `i % cols` 计算行列位置将标签添加到网格布局中；最后将完整的网格布局应用到UI界面的停车区域，如图10所示：

文本

AI 生成的内容可能不正确。

图10 动态生成停车位代码实现图

停车进场分为两种情况，一种是当停车场为空时（`t->length == 0`），代码调用 `carEnter` 函数让车辆进入停车场并分配车位，然后搜索该车辆获取其停车位置，在消息中记录进入时间、车牌号和车位号，接着生成10到30秒的随机时间创建单次触发定时器，定时器超时时会自动设置车牌号并调用出车函数，同时将定时器存储到 `carTimers` 映射表中并启动；当停车场已满时（`t->length == t->size`），先搜索该车辆是否已在停车场中，如果不在则检查等待队列是否已满，队列未满时会根据队列是否为空进行不同处理：队列为空时提示停车场已满并让车辆进入等待队列，队列不为空时会搜索车辆是否已在队列中，不在则入队，已存在则提示车辆已在等候区，而如果车辆已在停车场中则提示该车辆已存在，如图11所示：

文本

AI 生成的内容可能不正确。

图11 停车场进车代码实现图

超时自动离开功能中，函数 `checkQueueTimeout` 首先检查队列是否为空，如果为空则直接返回，否则获取当前时间并进入循环检查；在循环中取出队首车辆的进入时间字符串，将其转换为 `QTime` 对象，如果时间有效且与当前时间的间隔超过10秒，则获取该车辆的车牌号，在消息中记录超时离开的信息，调用 `DeQueue` 函数将车辆移出队列，并调用 `updateQueueIcons` 更新队列图标显示；如果队首车辆未超时则直接跳出循环（因为队列是按时间顺序排列的，队首未超时意味着后面的车辆也不会超时），最后将更新后的消息显示在界面上，如图12所示：

文本

AI 生成的内容可能不正确。

图12 等候区超时离开功能代码实现图

函数 `on\_checkbutton\_clicked` 首先计算停车场剩余车位数（总容量减去当前车辆数）和等待队列剩余位置数（最大容量减去当前队列长度），将这些信息添加到消息中；接着获取输入框中的车牌号，如果车牌号为空则提示用户输入车牌号，如果停车场为空则提示停车场为空，如果停车场有车辆则调用 `search` 函数查找该车牌号，找到时将车牌号、停车位位置和进入时间添加到消息中，未找到时提示车辆不存在；最后将完整的消息显示在界面的消息显示区域中，如图13所示：

文本

AI 生成的内容可能不正确。

图13 查询功能代码实现图

更新等候区图标中，函数 `updateQueueIcons` 首先遍历从1到 `Max` 的所有队列位置，通过 `findChild` 查找名为"queue1"、"queue2"等的 `QLabel` 控件，如果找到则将其图标设置为空车位图片以清空所有队列图标；接着遍历当前队列中的实际车辆数量，通过相同方式查找对应的 `QLabel` 控件，如果找到则将其图标设置为车辆图片，从而实现队列图标的动态更新显示，如图14所示：

图形用户界面, 文本, 应用程序

AI 生成的内容可能不正确。

图14 更新等候区图标代码实现图

计算停车费功能依靠函数 `calculate` ，接收进入时间和离开时间两个字符串参数，将它们分别转换为 `QDateTime` 对象，然后通过 `secsTo` 方法计算两个时间之间的秒数差值；根据停车时长采用分段计费：5秒及以内免费返回0元，5秒到20秒之间收费2元，20秒到60秒之间收费5元，超过60秒收费10元，如果不符合任何条件则默认返回0元，如图15所示：

文本

AI 生成的内容可能不正确。

图15 计算停车费功能代码实现图

随机分配车位功能函数 `createPos` 采用了"随机优先，顺序备选"的车位分配策略：首先使用 `rand()%max + 1` 生成1到最大车位数范围内的随机整数作为候选车位编号，然后检查该随机位置在 `position` 数组中对应索引（车位编号减1）的值是否为0（表示空闲状态），如果该位置空闲则立即将其标记为1（表示占用）并返回车位编号；当随机选中的车位已被占用时，函数转为顺序分配模式，从数组索引0开始逐一检查每个车位的占用状态，一旦发现第一个值为0的位置就将其标记为占用状态并返回实际车位编号（数组索引加1），这种设计既保证了车位分配的随机性以避免集中停放，又确保了在随机分配失败时能够找到任何可用车位，如果遍历完所有车位都没有找到空闲位置则返回-1表示停车场已满无法分配。

具体实现如图16所示：

文本

AI 生成的内容可能不正确。

图15 分配停车位功能代码实现图

等候区初始化函数 `createQueue` 使用 `std::make\_unique<queueNode>()` 创建一个智能指针管理的队列节点对象，将队列的容量设置为预定义的常量 `Max`，然后通过 `std::make\_unique<qcar[]>(q->capacity)` 为队列分配一个容量为 `Max` 的 `qcar` 类型数组用于存储等待车辆信息，接着调用 `MakeEmpty(q)` 函数对队列进行初始化（设置队首、队尾指针和大小等），最后返回这个完全初始化的队列智能指针对象。如图15所示：

文本

AI 生成的内容可能不正确。

图15 等候区初始化功能代码实现图

这个 `onTimeout()` 方法实现了停车场客户端的自动车辆进出模拟功能。方法首先通过随机数生成器决定是进车还是出车，其中进车的概率约为2/3（当随机数不等于0时），出车概率约为1/3。当决定进车或者停车场为空时，程序会生成一个随机的车牌号（包含省份简称、字母和数字），将其添加到本地的车辆集合 `carSet` 中，然后向服务器发送格式为 "IN:车牌号" 的消息。

当决定让车辆离开且停车场中有车时，程序会从 `carSet` 中随机选择一个已存在的车牌号。通过生成随机索引并使用迭代器定位到对应车牌，然后向服务器发送格式为 "OUT:车牌号" 的消息，同时从本地集合中移除该车牌。整个过程中，程序会通过 `qDebug()` 输出相应的日志信息，显示车辆的进入和离开状态，这样就实现了停车场车辆流动的自动化模拟。如图16所示：

文本

AI 生成的内容可能不正确。

图16 客户端进出车功能代码实现图

等候区图标更新功能具体流程是：遍历等待队列中的所有车辆，为每辆车计算其在队列中的位置（从1开始编号），然后通过 `findChild` 方法找到对应的 QLabel 控件（如 queue1、queue2 等），最后将车辆图片（car.png）设置到这些标签上，让用户能够直观地看到等待队列中有哪些位置被车辆占用。

这样用户就能在界面上清楚地看到等待队列的状态，知道有多少车在排队等待进入停车场。

**五、系统实现与测试**

先运行服务端程序，再运行客户端程序，debug()函数在终端打印日志。

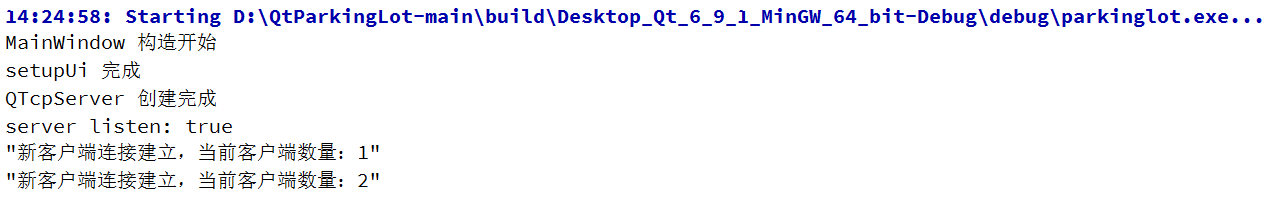


图17 服务端日志图

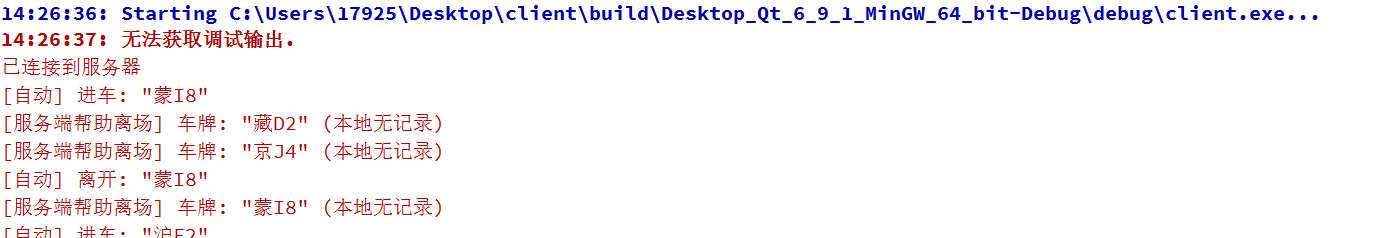


图18 客户端日志图

这时候程序的主界面如图19所示：

图形用户界面

AI 生成的内容可能不正确。

图19 程序主界面图

超时自动离场功能如图20所示：

图形用户界面

AI 生成的内容可能不正确。

图20 超时自动离场功能图

查询当前信息功能如图21所示：

图形用户界面, 应用程序

AI 生成的内容可能不正确。

图21 查询当前信息功能图

**六、总结与分析**

服务端层面上，是一个基于Qt框架开发的智能化停车场管理应用程序，采用C++语言实现，具备完整的图形用户界面和网络通信功能。系统集成了车辆进出管理、等待队列处理、费用计算、实时监控等核心功能，是一个较为完整的物联网应用实例。

系统采用模块化设计思想，主要分为以下几个核心模块：首先是用户界面模块，基于Qt的MainWindow类实现，提供直观的图形化操作界面；其次是数据管理模块，包含停车场表结构（table）和等待队列结构（queue），使用智能指针进行内存管理，体现了现代C++的编程理念；第三是网络通信模块，通过QTcpServer实现TCP服务端功能，支持多客户端并发连接，为物联网应用奠定基础；最后是定时器模块，实现车辆自动离场和队列超时检测等智能化功能。

系统在数据结构设计方面表现出较强的专业性。停车场采用线性表结构存储车辆信息，包含车位编号、车牌号和进入时间等关键数据；等待队列采用循环队列结构，最大容量为5辆车，有效解决了停车场满员时的排队问题。两种数据结构都使用了C++11的智能指针技术，避免了内存泄漏问题，提高了程序的稳定性和安全性。

系统具备多项实用功能特性：动态停车位生成功能可根据需求灵活调整停车场规模；随机车牌号生成功能便于系统测试和演示；智能费用计算系统根据停车时长自动计算费用，体现了商业化应用的考虑；TCP网络通信功能支持远程车辆进出控制，为物联网集成提供了接口；定时器机制实现了车辆自动离场和队列超时处理，提升了系统的自动化水平。

该系统在物联网应用方面具有良好的扩展性和实用性。TCP服务器功能为车牌识别摄像头、地磁传感器等硬件设备的接入提供了标准化接口；日志记录功能支持系统运行状态的实时监控和历史数据分析；模块化的代码结构便于后续功能扩展，如移动支付接口、车位预约系统等高级功能的集成。

而客户端程序采用Qt框架开发，实现了一个智能停车场管理系统的车辆终端模拟器。程序通过TCP协议与服务器建立长连接，模拟真实停车场环境中车辆的进出场景。

客户端程序采用面向对象的设计模式，核心Client类继承自QObject，充分利用了Qt的信号槽机制实现异步事件处理。程序通过QTcpSocket类建立与服务器的网络连接，使用QTimer类实现定时自动化操作，整体架构简洁高效，符合物联网设备轻量化的设计理念。

程序的核心功能包括随机车牌号生成、车辆进出场模拟和状态管理。车牌号生成算法覆盖了全国主要省份简称，采用"省份+字母+数字"的标准格式，提高了模拟数据的真实性。车辆进出逻辑通过概率算法控制，当停车场为空时强制进车，否则随机决定进出操作，有效模拟了实际停车场的使用场景。

客户端与服务器之间采用简洁的文本协议进行通信，使用"IN:车牌号"和"OUT:车牌号"格式发送车辆进出信息，协议简单易解析，降低了网络传输开销。同时支持接收服务器的"HELP\_OUT:车牌号"指令，实现了双向通信和远程控制功能，体现了物联网系统的互操作性。

程序使用QSet容器维护本地车辆状态，确保了车辆进出的逻辑一致性。通过本地状态缓存机制，客户端能够准确跟踪当前停车场内的车辆情况，避免了重复进出等异常操作，保证了系统的可靠性和数据完整性。

客户端具备全自动运行能力，通过2秒定时器实现连续的车辆进出模拟，无需人工干预即可长时间稳定运行。这种设计特别适合物联网测试环境，能够有效验证服务器的并发处理能力和系统稳定性，为实际部署提供了有力的技术支撑。

整个客户端程序体现了现代物联网设备的典型特征：轻量化架构、标准化通信、自动化运行和智能化管理。程序代码结构清晰，异常处理完善，具有良好的可扩展性和维护性，为停车场物联网系统的实际应用奠定了坚实的技术基础。

系统在技术实现上体现了多个优势：采用现代C++特性如智能指针和lambda表达式，提高了代码质量；Qt框架的跨平台特性保证了系统的可移植性；网络编程的集成为系统向物联网方向发展奠定了基础；用户界面设计直观友好，降低了操作门槛。特别值得注意的是，系统考虑了多线程并发的复杂性，选择了单线程事件驱动模型，在保证功能完整性的同时避免了锁机制的复杂性。

通过以上分析可以看出，该停车场管理系统不仅实现了基本的停车场管理功能，更重要的是体现了现代软件工程的设计理念和物联网应用的发展趋势，是一个具有较高技术水平和实用价值的综合性项目。

**七、附录**

服务端代码：

table.h

#ifndef TABLE\_H

#define TABLE\_H

#include <stdio.h>

#include <memory>

#include "mainwindow.h"

#define COL 5 // 停车场每行车位数

#define ROW 2 // 停车场行数

// 停车场中车辆信息结构体

struct car

{

    int pos;           // 停车位编号

    QString num;       // 车牌号

    QString entime;    // 进入停车场时间

};

// 停车场表结构体

struct tableNode

{

    int length, size;              // length: 当前车辆数，size: 停车场总车位数

    std::unique\_ptr<car[]> cars;   // 存储车辆的数组

};

using table = std::unique\_ptr<tableNode>; // 停车场类型定义

table createTable(int); // 创建一个停车场表

int IsEmpty(const table& t); // 检测停车场是否为空

int IsFull(const table& t);  // 检测停车场是否满

void MakeEmpty(table& t);    // 将停车场表置空

void carEnter(table& t, const QString& num, const QString& etime, int pos); // 车辆进入停车场

void carOut(table& t, const QString& num); // 车辆离开停车场

int search(const table& t, const QString& num); // 查找停车场中是否有指定车牌号的车辆

float calculate(const QString& etime, const QString& outtime); // 计算停车费用

int createPos(int position[], int); // 随机分配停车位

#endif // TABLE\_H

queue.h

#ifndef QUEUE\_H

#define QUEUE\_H

#include <stdio.h>

#include <memory>

#include "mainwindow.h"

#define Max 5 // 等待队列最大容量

// 队列中车辆信息结构体

struct qcar

{

    int pos;           // 停车位编号

    QString num;       // 车牌号

    QString enterTime; // 进入队列的时间

};

// 循环队列结构体

struct queueNode{

    int capacity;                // 队列容量

    int front;                   // 队首指针

    int rear;                    // 队尾指针

    int size;                    // 当前队列元素个数

    std::unique\_ptr<qcar[]> wait; // 存储车辆的数组

};

using queue = std::unique\_ptr<queueNode>; // 队列类型定义

int IsEmpty(const queue& q); // 检测队列是否为空

int IsFull(const queue& q);  // 检测队列是否满

queue createQueue();         // 创建一个队列

void MakeEmpty(queue& q);    // 将队列置空

void enter(queue& q, const QString& num, const QString& time); // 车辆进队列

qcar Front(const queue& q);  // 返回队首元素

void DeQueue(queue& q);      // 队首出队

qcar FrontAndDeQueue(queue& q); // 返回队首并出队

int search(const queue& q, const QString& num); // 查找队列中是否有指定车牌号的车辆

#endif // QUEUE\_H

mainwindow.h

#ifndef MAINWINDOW\_H

#define MAINWINDOW\_H

#include <QMainWindow>

#include <QTimer>

#include <QTcpServer>

#include <QTcpSocket>

#include <QList>

#include <QMap>

QT\_BEGIN\_NAMESPACE

namespace Ui { class MainWindow; }

QT\_END\_NAMESPACE

// 主窗口类，负责停车场管理系统的主要逻辑和界面

class MainWindow : public QMainWindow

{

    Q\_OBJECT

public:

    MainWindow(QWidget \*parent = nullptr);

    ~MainWindow();

private slots:

    void updateQueueIcons(); // 更新等待队列图标

    void on\_generateLicenseButton\_clicked(); // 生成随机车牌号按钮点击事件

    QString generateLicensePlate(); // 生成随机车牌号

    void generateParkingSlots(int); // 动态生成停车位

    void on\_generateButton\_clicked(); // 生成停车位按钮点击事件

    void clearLayout(QLayout\*); // 清空布局

    void on\_enterbutton\_clicked();// 进车按钮点击事件

    void on\_outbutton\_clicked();// 出车按钮点击事件

    void on\_checkbutton\_clicked();// 查询信息按钮点击事件

    void checkQueueTimeout(); // 检查队列超时槽函数

protected:

    Ui::MainWindow \*ui; // 主界面指针

    QTimer\* queueTimer; // 队列超时检测定时器

    QTcpServer\* server; // TCP服务端监听对象

    QList<QTcpSocket\*> clients; // 已连接的客户端列表

    QMap<QString, QTimer\*> carTimers; // 车牌号到定时器的映射，用于自动离场

};

#endif // MAINWINDOW\_H

tablefunction.cpp

#include <stdio.h>

#include "table.h"

#include <stdlib.h>

#include <QCoreApplication>

#include <iostream>

#include <QDateTime>

// 判断停车场是否为空

int IsEmpty(const table& t){

    return t == nullptr || t->length == 0;

}

// 判断停车场是否已满

int IsFull(const table& t){

    return t != nullptr && t->length == t->size;

}

// 停车场置空

void MakeEmpty(table& t){

    if (t) t->length = 0;

}

// 创建一个停车场表，分配空间并初始化

// 参数：num 停车位总数

// 返回：新建的停车场表

table createTable(int num){

    auto t = std::make\_unique<tableNode>();

    t->size = num;

    t->cars = std::make\_unique<car[]>(t->size);

    MakeEmpty(t);

    return t;

}

// 车辆进入停车场

// 参数：t 停车场表，num 车牌号，etime 进入时间，pos 车位编号

void carEnter(table& t, const QString& num, const QString& etime, int pos){

    if(IsFull(t))

       printf("Parkinglot is Full!\n");

    else{

        t->cars[t->length].num = num;

        t->cars[t->length].entime = etime;

        t->cars[t->length].pos = pos;

        t->length++;

    }

}

// 车辆离开停车场

// 参数：t 停车场表，num 车牌号

void carOut(table& t, const QString& num){

    int index = search(t, num);

    if(index == -1)

       printf("not found!!!\n");

    else{

        for(int i = index; i < t->length - 1; i++){

            t->cars[i] = t->cars[i+1];

        }

        t->length--;

    }

}

// 查找停车场中是否有指定车牌号的车辆

// 返回下标，未找到返回-1

int search(const table& t, const QString& num){

    int sign = -1;

    if(IsEmpty(t))

      printf("empty table!\n");

    else{

        int n = t->length;

        for(int i = 0; i < n; i++){

            if(t->cars[i].num == num){

                printf("found it!!!\n");

                sign = i;

                break;

            }

        }

    }

    return sign;

}

// 计算停车费用

// 参数：etime 进入时间，outtime 离开时间

// 返回：费用（元）

float calculate(const QString& etime, const QString& outtime){

    QDateTime timeen, timeout;

    timeen = QDateTime::fromString(etime, "hh:mm:ss");

    timeout = QDateTime::fromString(outtime, "hh:mm:ss");

    int second = timeen.secsTo(timeout);

    if(second <= 5)

        return 0;

    else if(second > 5 && second <= 20)

        return 2;

    else if(second > 20 && second <= 60)

        return 5;

    else if(second > 60)

        return 10;

    return 0;

}

// 随机分配停车位

// 参数：position 停车位占用数组，maxnum 停车位总数

// 返回：分配到的车位编号

int createPos(int position[], int maxnum){

    int max = maxnum, pos = rand()%max + 1;

    if(position[pos-1] == 0){

        position[pos-1] = 1;

        return pos;

    }

    else{

        int i;

        for (i = 0; i < max; i++)

        {

            if(position[i] == 0){

                position[i] = 1;

                return i+1;

            }

        }

    }

    return -1;

}

queuefunction.cpp

#include <stdio.h>

#include "queue.h"

#include <stdlib.h>

// 判断队列是否为空

int IsEmpty(const queue& q){

   return q == nullptr || q->size == 0;

}

// 判断队列是否已满

int IsFull(const queue& q){

    return q != nullptr && q->size == q->capacity;

}

// 队列初始化/置空

void MakeEmpty(queue& q){

    if (q) {

        q->rear = 0;

        q->front = 0;

        q->size = 0;

    }

}

// 创建一个队列，分配空间并初始化

queue createQueue(){

    auto q = std::make\_unique<queueNode>();

    q->capacity = Max;

    q->wait = std::make\_unique<qcar[]>(q->capacity);

    MakeEmpty(q);

    return q;

}

// 车辆进队列

// 参数：q 队列，num 车牌号，time 进入时间

void enter(queue& q, const QString& num, const QString& time){

    if(IsFull(q))

       printf("Full queue\n");

    else{

       q->wait[q->rear].num = num;

       q->wait[q->rear].enterTime = time;

       q->size++;

       if(++q->rear == q->capacity)

          q->rear = 0; // 循环队列处理

    }

}

// 获取队首元素

qcar Front(const queue& q){

    return q->wait[q->front];

}

// 队首出队

void DeQueue(queue& q){

    if(IsEmpty(q))

        printf("empty queue!\n");

    else{

        q->size--;

        q->front++;

        if(q->front == q->capacity)

            q->front = 0; // 循环队列处理

    }

}

// 返回队首并出队

qcar FrontAndDeQueue(queue& q){

    if(!IsEmpty(q))

        return q->wait[q->front++];

    else{

        printf("empty queue!\n");

        return qcar{};

    }

}

// 查找队列中是否有指定车牌号的车辆

// 返回下标，未找到返回-1

int search(const queue& q, const QString& num){

    int sign = -1;

    if(IsEmpty(q))

      printf("empty table!\n");

    else{

        int n = q->size;

        for(int i = 0; i < n; i++){

            if(q->wait[i].num == num){

                printf("found it!!!\n");

                sign = i;

                break;

            }

        }

    }

    return sign;

}

mainwindow.cpp

#include "mainwindow.h"

#include "ui\_mainwindow.h"

#include "queue.h"

#include "table.h"

#include "queuefunction.cpp"

#include "tablefunction.cpp"

#include <QTime>

#include <QRandomGenerator>

#include <QStringList>

#include <QTimer>

#include <QDebug>

#include <QTcpServer>

#include <QTcpSocket>

#include <QFile>

#include <QTextStream>

// 6.16版本

// 为什么不用多线程：锁粒度大，哈希分片要负载均衡，cas操作麻烦，日志需要双缓冲区

// 用智能指针管理table和queue

// t：停车场表，q：等待队列

// message：用于显示信息，waits：等待队列车牌号数组

// position：停车位占用情况，queueposition：队列占用情况，p：临时变量

// carTimers：每辆车的自动离场定时器

table t;

queue q = createQueue();

QString message, waits[Max];

int position[COL\*ROW] = {0}, queueposition[Max] = {0}, p;

QMap<QString, QTimer\*> carTimers;

// 日志记录函数，将信息写入 debuglog.txt

void myLog(const QString &msg) {

    QFile file("D:/QtParkingLot-main/debuglog.txt");

    if (file.open(QIODevice::Append | QIODevice::Text)) {

        QTextStream out(&file);

        out << msg << "\n";

        file.close();

    }

}

// 主窗口构造函数，初始化界面、定时器、TCP服务端等

MainWindow::MainWindow(QWidget \*parent)

    : QMainWindow(parent)

    , ui(new Ui::MainWindow)

{

    qDebug() << "MainWindow 构造开始";

    ui->setupUi(this);

    qDebug() << "setupUi 完成";

    queueTimer = new QTimer(this);

    // connect(发送者, &发送者::信号, 接收者, &接收者::槽函数);

    // connect(queueTimer, &QTimer::timeout, this, &MainWindow::checkQueueTimeout);

    connect(queueTimer, &QTimer::timeout, this, [=](){

        checkQueueTimeout();

    });

    queueTimer->start(1000); // 每秒检查一次队列超时

    // 初始化服务端监听

    server = new QTcpServer(this);

    qDebug() << "QTcpServer 创建完成";

    // 第一遍connect：监听服务器是否有新客户端连接。

    // 第二遍connect：为每个新连接的客户端socket设置数据和断开事件的处理。

    connect(server, &QTcpServer::newConnection, this, [=](){

        while (server->hasPendingConnections()) {

            QTcpSocket\* client = server->nextPendingConnection();

            clients.append(client);

            // 处理客户端消息

            connect(client, &QTcpSocket::readyRead, this, [=](){

                // 当客户端有数据可读时，循环读取每一行数据

                while (client->bytesAvailable()) {

                    // 读取客户端发来的一行数据

                    QByteArray data = client->readLine();

                    // 将读取到的数据转换为QString，并去除首尾空白字符

                    QString msg = QString::fromUtf8(data).trimmed();

                    // 判断消息类型

                    if (msg.startsWith("IN:")) {

                        // 如果消息以IN:开头，表示有车辆要进入停车场

                        // 提取车牌号

                        QString plate = msg.section(':', 1, 1);

                        // 设置车牌号到输入框

                        ui->inputnum->setText(plate);

                        // 自动调用进车按钮的处理函数，实现车辆入场

                        on\_enterbutton\_clicked();

                    } else if (msg.startsWith("OUT:")) {

                        // 如果消息以OUT:开头，表示有车辆要离开停车场

                        // 提取车牌号

                        QString plate = msg.section(':', 1, 1);

                        // 设置车牌号到输入框

                        ui->inputnum->setText(plate);

                        // 自动调用出车按钮的处理函数，实现车辆出场

                        on\_outbutton\_clicked();

                    }

                }

            });

            // 客户端断开连接处理

            connect(client, &QTcpSocket::disconnected, this, [=](){

                clients.removeAll(client);

                client->deleteLater();

            });

        }

        int clientCountAfter = clients.size();

        myLog(QString("新客户端连接建立，当前客户端数量：%1").arg(clientCountAfter));

        qDebug() << QString("新客户端连接建立，当前客户端数量：%1").arg(clientCountAfter);

    });

    server->listen(QHostAddress::Any, 12345); // 监听端口

    qDebug() << "server listen:" << server->isListening();

}

// 析构函数，释放UI资源

MainWindow::~MainWindow()

{

    delete ui;

}

// 生成随机车牌号

QString MainWindow::generateLicensePlate() {

    // 省份代码

    QStringList provinces = { "京", "沪", "川", "渝", "鄂", "粤", "鲁", "晋", "蒙", "桂", "甘", "贵", "黑", "吉", "辽", "闽", "赣", "闽", "青", "琼", "藏", "新", "苏" };

    QString province = provinces[QRandomGenerator::global()->bounded(provinces.size())]; // 随机选择一个省份

    QString letter = QString(QChar('A' + QRandomGenerator::global()->bounded(10))); // 随机选择一个大写字母

    QString numberPart;

    numberPart.append(QString::number(QRandomGenerator::global()->bounded(10)));

    // 可扩展为五位字符组合

    return province + letter + numberPart; // 返回完整车牌号

}

// 生成车牌号按钮点击事件

void MainWindow::on\_generateLicenseButton\_clicked() {

    QString licensePlate = generateLicensePlate(); // 调用生成车牌号函数

    ui->inputnum->setText(licensePlate); // 将车牌号填充到 inputnum 输入框

}

// 动态生成停车位

void MainWindow::generateParkingSlots(int numSlots)

{

    // 1. 清空当前布局，防止重复添加

    QLayout \*currentLayout = ui->parkingArea->layout();

    if (currentLayout != nullptr) {

        QLayoutItem \*item;

        while ((item = currentLayout->takeAt(0)) != nullptr) {

            delete item->widget(); // 删除布局中的所有控件

            delete item; // 删除布局项

        }

        delete currentLayout; // 删除旧布局

    }

    // 2. 创建一个新的 QGridLayout 布局

    QGridLayout \*parkingLayout = new QGridLayout();

    // 3. 动态生成车位

    int rows = 5;  // 每行5个车位

    int cols = (numSlots + rows - 1) / rows;  // 计算列数，确保合理分布车位

    t = createTable(numSlots);

    for (int i = 0; i < numSlots; ++i) {

        // 创建一个车位标签 (QLabel)

        QLabel \*slotLabel = new QLabel("parkinglot"+ QString::number(i + 1), this);

        // 为车位设置图片 (可以是空车位图)

        slotLabel->setPixmap(QPixmap("D:/QtParkingLot-main/parkinglot.png"));

        slotLabel->setAlignment(Qt::AlignCenter); // 设置图片居中

        // 设置车位标签的大小

        slotLabel->setFixedSize(50, 50); // 设置每个车位的尺寸

        // 4. 将车位标签添加到 QGridLayout 中

        parkingLayout->addWidget(slotLabel, i / cols, i % cols); // 按行列布局

    }

    // 5. 将 QGridLayout 布局应用到停车区域

    ui->parkingArea->setLayout(parkingLayout);

}

// 生成停车位按钮点击事件

void MainWindow::on\_generateButton\_clicked()

{

    // 获取用户输入的车位数量

    int numSlots = ui->inputParkingSlots->text().toInt();

    // 确保输入为正整数

    if (numSlots > 0) {

        // 调用函数生成车位

        generateParkingSlots(numSlots);

    }

}

// 清空布局，释放控件

void MainWindow::clearLayout(QLayout \*layout)

{

    while (QLayoutItem \*item = layout->takeAt(0)) {

        delete item->widget(); // 删除控件

        delete item; // 删除布局项

    }

}

// 进车按钮点击事件，处理车辆进入停车场或等待队列的逻辑

void MainWindow::on\_enterbutton\_clicked()

{

    QString num = ui->inputnum->text(),etime;

    QTime current\_time = QTime::currentTime();

    etime = current\_time.toString("hh:mm:ss");//获取当前时间

    //如果输入为空且队列为空

    if(num == "" && q->size == 0){

        message.append("请输入车牌号！\n");

    }

    //第一次输入车牌号，进停车场

    else if(t->length == 0){

        carEnter(t,num,etime,createPos(position,t->size));

        int index = search(t, num);

        int position = t->cars[index].pos;

        message.append(etime + ": " + "车牌号为"+num + "进入停车场\n"+"停在了车位："+QString::number(position)+"\n");

        // 启动自动离开定时器

        int randomSeconds = QRandomGenerator::global()->bounded(10, 31); // 10~30秒

        QTimer\* timer = new QTimer(this);

        timer->setSingleShot(true);

        connect(timer, &QTimer::timeout, this, [=]() {

            ui->inputnum->setText(num);

            on\_outbutton\_clicked();

            carTimers.remove(num);

            timer->deleteLater();

        });

        carTimers[num] = timer;

        timer->start(randomSeconds \* 1000);

    }

    //当停车场为满时

    else if(t->length == t->size){

        int index = search(t,num);//输入车牌号，查找停车场中是否有这辆车

        if(index == -1){

            if(IsFull(q))//若队列为满，不进行入队并提示

                message.append("等待队列已满！\n");

            else {

                if(q->size == 0){//第一次往队列中进车

                    message.append("停车场已满！\n");

                    enter(q,num,etime);

                    message.append(etime + ": " + "车牌号为"+ num + "进入等待队列\n");

                }

                else{

                    int index = search(q,num);//查找队列中是否有这辆车

                    if(index == -1){//若没有，则入队列

                        enter(q,num,etime);

                        message.append(etime + ": " + "车牌号为"+ num + "进入等待队列\n");

                    }

                    else{//若有，则重新输入

                        message.append("该车辆已存在等候区！\n");

                    }

                }

            }

        }

        else

            message.append("该车辆已存在停车场！请重新输入。\n");

    }

    else if(t->length >= 0 && q->size > 0){

        //停车场有空位且队列不为空（防御性编程，健壮性）

        //车牌不为空时，队列首位车入停车场，输入的车入队列

        if(t->length < t->size){

            if(num == ""){

                message.append("请输入车牌号！\n");

            }

            else{

                int index = search(t,num);//输入车牌号，查找停车场中是否有该车

                if(index == -1){

                    if(IsFull(q))

                        message.append("等待队列已满！\n");

                    else {

                        if(q->size == 0){

                            message.append("停车场已满！\n");

                            enter(q,num,etime);

                            message.append(etime + ": " + "车牌号为"+ num + "进入等待队列\n");

                        }

                        else{

                            int index = search(q,num);

                            if(index == -1){

                                enter(q,num,etime);

                                message.append(etime + ": " + "车牌号为"+ num + "进入等待队列\n");

                            }

                            else{

                                message.append("该车辆已存在等待区！请重新输入。\n");

                            }

                        }

                    }

                }

                else{

                    message.append("该车辆已存在停车场！请重新输入。\n");

                }

            }

        }

        int index = search(t,num);

        if(index >= 0){

            message.append("该车辆已存在！请重新输入。\n");

        }

        else{

            QString tempnum = q->wait[q->front].num;

            //出车后，复原队列车位图标

            carEnter(t,tempnum,etime,createPos(position,t->size));

            switch (q->size) {

            case 1:

                ui->queue1->setText("1");

                break;

            case 2:

                ui->queue2->setText("2");

                break;

            case 3:

                ui->queue3->setText("3");

                break;

            case 4:

                ui->queue4->setText("4");

                break;

            case 5:

                ui->queue5->setText("5");

                break;

            }

            DeQueue(q);//队首入停车场，出队列

            message.append(etime + ": " + tempnum + "进入停车场\n");

            // 启动自动离开定时器

            int randomSeconds = QRandomGenerator::global()->bounded(10, 31); // 10~30秒

            QTimer\* timer = new QTimer(this);

            timer->setSingleShot(true);

            connect(timer, &QTimer::timeout, this, [=]() {

                ui->inputnum->setText(tempnum);

                on\_outbutton\_clicked();

                carTimers.remove(tempnum);

                timer->deleteLater();

            });

            carTimers[tempnum] = timer;

            timer->start(randomSeconds \* 1000);

        }

    }

    //若停车场不满，获取输入车牌号并进行判断，若无该车，则入停车场

    else if(t->length > 0){

        int index = search(t,num);

        if(index >= 0){

            message.append("该车辆已存在！请重新输入。\n");

        }

        else{

            carEnter(t,num,etime,createPos(position,t->size));

            int index = search(t, num);

            int position = t->cars[index].pos;

            message.append(etime + ": " + "车牌号为"+ num + "进入停车场\n"+"停在了车位："+QString::number(position)+"\n");

            // 启动自动离开定时器

            int randomSeconds = QRandomGenerator::global()->bounded(10, 31); // 10~30秒

            QTimer\* timer = new QTimer(this);

            timer->setSingleShot(true);

            connect(timer, &QTimer::timeout, this, [=]() {

                ui->inputnum->setText(num);

                on\_outbutton\_clicked();

                carTimers.remove(num);

                timer->deleteLater();

            });

            carTimers[num] = timer;

            timer->start(randomSeconds \* 1000);

        }

    }

    // 更新停车场车位图标

    for (int i = 0; i < t->length; i++) {

        int pos = t->cars[i].pos - 1; // 车位位置，从0开始

        if (pos >= 0 && pos < t->size) { // 确保位置在有效范围内

            QString imagePath = "D:/QtParkingLot-main/car.png";

            // 从 parkingArea 中查找对应的车位标签

            // 把停车区里所有的车位标签（QLabel）都找出来，放到一个列表里

            QList<QLabel\*> slotLabels = ui->parkingArea->findChildren<QLabel\*>();

            // 防止越界

            if (slotLabels.size() > pos) {

                // 取出第pos个车位标签

                QLabel\* slotLabel = slotLabels.at(pos);

                slotLabel->setPixmap(QPixmap(imagePath)); // 设置车位为占用

            }

        }

    }

    // 更新等待队列的图标

    for (int i = 0; i < q->size; i++) {

        int queueIndex = i + 1;

        QString imagePath = "D:/QtParkingLot-main/car.png";

        QLabel\* queueLabel = findChild<QLabel\*>(QString("queue%1").arg(queueIndex));

        if (queueLabel) {

            queueLabel->setPixmap(QPixmap(imagePath)); // 设置队列的车辆图标

        }

    }

    //在控制台输出信息

    ui->showmessage->setText(message);

}

// 出车按钮点击事件，处理车辆离开停车场的逻辑

void MainWindow::on\_outbutton\_clicked()

{

    QTime current\_time = QTime::currentTime();

    QString outtime = current\_time.toString("hh:mm:ss"), num = ui->inputnum->text(); // 获取出停车场时间

    int index = search(t, num);

    if (t->length == 0) {

        message.append("停车场为空！\n");

    }

    else if (index > -1) {

        // 释放车位

        position[t->cars[index].pos - 1] = 0;

        // 查找并重置停车位图标

        int pos = t->cars[index].pos - 1; // 获取车位的索引

        if (pos >= 0 && pos < t->size) {

            QString imagePath ="D:/QtParkingLot-main/parkinglot.png"; // 空车位的图片路径

            // 从 parkingArea 中查找对应的车位标签

            QList<QLabel\*> slotLabels = ui->parkingArea->findChildren<QLabel\*>();

            if (slotLabels.size() > pos) {

                QLabel\* slotLabel = slotLabels.at(pos);

                slotLabel->setPixmap(QPixmap(imagePath)); // 设置车位为空

            }

        }

        // 车辆出停车场

        carOut(t, num);

        // 移除并销毁定时器

        if (carTimers.contains(num)) {

            carTimers[num]->stop();

            carTimers[num]->deleteLater();

            carTimers.remove(num);

        }

        // 新增：通知所有客户端帮助离场

        for (QTcpSocket\* client : clients) {

            if (client && client->state() == QAbstractSocket::ConnectedState) {

                client->write(QString("HELP\_OUT:%1\n").arg(num).toUtf8());

            }

        }

        // 计算停车费用

        float cost = calculate(t->cars[index].entime, outtime);

        QString qcost = QString::asprintf("%.2f", cost);

        message.append(outtime + ": " + num + "退出停车场\n");

        message.append("收费: " + qcost + "元\n");

        // 如果队列中有车，首辆车进入停车场

        if (!IsEmpty(q)) {

            QString tempnum = q->wait[q->front].num;

            carEnter(t, tempnum, outtime, createPos(position, t->size));

            message.append(outtime + ": " + tempnum + "从等待队列进入停车场\n");

            // 找到该车停放的位置

            int newPos = t->cars[search(t, tempnum)].pos - 1; // 获取新停放位置索引

            if (newPos >= 0 && newPos < t->size) {

                QString imagePath = "D:/QtParkingLot-main/car.png";

                QList<QLabel\*> slotLabels = ui->parkingArea->findChildren<QLabel\*>();

                if (slotLabels.size() > newPos) {

                    QLabel\* slotLabel = slotLabels.at(newPos);

                    slotLabel->setPixmap(QPixmap(imagePath)); // 更新车位图标为已占用

                }

            }

            // 出队列并更新队列图标

            DeQueue(q);

            updateQueueIcons();

            // 启动自动离开定时器

            int randomSeconds = QRandomGenerator::global()->bounded(10, 31); // 10~30秒

            QTimer\* timer = new QTimer(this);

            timer->setSingleShot(true);

            connect(timer, &QTimer::timeout, this, [=]() {

                ui->inputnum->setText(tempnum);

                on\_outbutton\_clicked();

                carTimers.remove(tempnum);

                timer->deleteLater();

            });

            carTimers[tempnum] = timer;

            timer->start(randomSeconds \* 1000);

    }

    }

    ui->showmessage->setText(message);

}

// 更新等待队列的图标显示

void MainWindow::updateQueueIcons()

{

    // 清空所有队列图标

    for (int i = 1; i <= Max; i++) {

        QLabel\* queueLabel = findChild<QLabel\*>(QString("queue%1").arg(i));

        if (queueLabel) {

            QString emptyPath = "D:/QtParkingLot-main/parkinglot.png";

            queueLabel->setPixmap(QPixmap(emptyPath));  // 队列图标清空

        }

    }

    // 重新设置队列图标

    for (int i = 0; i < q->size; i++) {

        int queueIndex = i + 1;

        QString imagePath = "D:/QtParkingLot-main/car.png";

        QLabel\* queueLabel = findChild<QLabel\*>(QString("queue%1").arg(queueIndex));

        if (queueLabel) {

            queueLabel->setPixmap(QPixmap(imagePath));  // 设置队列中的车辆图标

        }

    }

}

// 查询按钮点击事件，显示停车场和队列剩余情况及车辆信息

void MainWindow::on\_checkbutton\_clicked()

{

    QString qnum = QString::number(Max - q->size),tnum = QString::number(t->size - t->length);//计算停车场剩余位置和队列剩余位置

    message.append("停车场剩余" + tnum + "位," + "等候队列剩余" + qnum + "位。\n");

    QString num = ui->inputnum->text();

    if(num == ""){

        message.append("请输入车牌号！\n");

    }else if(t->length == 0){

        message.append("停车场为空！\n");

    }else if(t->length > 0){

        int index = search(t,num);

        if(index >= 0){

            QString pos = QString::number(t->cars[index].pos);

            message.append("车牌号：" + num + "停车位：" + pos + "进入时间：" + t->cars[index].entime + "\n");

        }else{

            message.append("车辆不存在！\n");

        }

    }

    ui->showmessage->setText(message);

}

// 检查等待队列超时，超时则自动离开

void MainWindow::checkQueueTimeout()

{

    if(q->size == 0) return;

    QTime now = QTime::currentTime();

    // 循环检查队首，只要超时就一直处理

    while (q->size > 0) {

        QString enterTime = q->wait[q->front].enterTime;

        QTime carTime = QTime::fromString(enterTime, "hh:mm:ss");

        if (carTime.isValid() && carTime.secsTo(now) >= 10) {

            QString carNum = q->wait[q->front].num;

            message.append(now.toString("hh:mm:ss") + ": " + "车牌号为" + carNum + "在队列中等待超过10秒，已自动离开\n");

            DeQueue(q);

            updateQueueIcons();

        } else {

            // 队首没超时，后面的车肯定也没超时，直接退出循环

            break;

        }

    }

    ui->showmessage->setText(message);

}

客户端：

main.cpp

#include <QCoreApplication>

#include <QTcpSocket>

#include <QTimer>

#include <QRandomGenerator>

#include <QTextStream>

#include <QSet>

#include <QDebug>

QString generateLicensePlate() {

    QStringList provinces = { "京", "沪", "川", "渝", "鄂", "粤", "鲁", "晋", "蒙", "桂", "甘", "贵", "黑", "吉", "辽", "闽", "赣", "闽", "青", "琼", "藏", "新", "苏" };

    QString province = provinces[QRandomGenerator::global()->bounded(provinces.size())];

    QString letter = QString(QChar('A' + QRandomGenerator::global()->bounded(10)));

    QString numberPart;

    numberPart.append(QString::number(QRandomGenerator::global()->bounded(10)));

    return province + letter + numberPart;

}

class Client : public QObject {

    Q\_OBJECT

public:

    Client(QObject\* parent = nullptr) : QObject(parent) {

        connect(&socket, &QTcpSocket::connected, this, &Client::onConnected);

        connect(&socket, &QTcpSocket::readyRead, this, &Client::onReadyRead);

        connect(&timer, &QTimer::timeout, this, &Client::onTimeout);

    }

    void start(const QString& host, quint16 port) {

        socket.connectToHost(host, port);

    }

public slots:

    void onConnected() {

        qDebug() << "已连接到服务器";

        timer.start(2000); // 每2秒自动进出车

    }

    void onTimeout() {

        // 随机决定进车还是离开车

        bool isEnter = (QRandomGenerator::global()->bounded(3) != 0);

        QString plate;

        if (isEnter || carSet.isEmpty()) {

            // 进车

            plate = generateLicensePlate();

            carSet.insert(plate);

            QString msg = "IN:" + plate + "\n";

            socket.write(msg.toUtf8());

            socket.flush();

            qDebug() << "[自动] 进车:" << plate;

        } else {

            // 离开车

            int idx = QRandomGenerator::global()->bounded(carSet.size());

            auto it = carSet.begin();

            std::advance(it, idx);

            plate = \*it;

            QString msg = "OUT:" + plate + "\n";

            socket.write(msg.toUtf8());

            socket.flush();

            carSet.remove(plate);

            qDebug() << "[自动] 离开:" << plate;

        }

    }

    void onReadyRead() {

        while (socket.bytesAvailable()) {

            QByteArray data = socket.readLine();

            QString msg = QString::fromUtf8(data).trimmed();

            if (msg.startsWith("HELP\_OUT:")) {

                QString plate = msg.section(':', 1, 1);

                if (carSet.contains(plate)) {

                    carSet.remove(plate);

                    qDebug() << "[服务端帮助离场] 车牌:" << plate;

                } else {

                    qDebug() << "[服务端帮助离场] 车牌:" << plate << "(本地无记录)";

                }

            } else {

                qDebug() << "[收到服务端消息]:" << msg;

            }

        }

    }

private:

    QTcpSocket socket;

    QTimer timer;

    QSet<QString> carSet; // 记录已进场的车牌

};

#include "main.moc"

int main(int argc, char \*argv[])

{

    QCoreApplication a(argc, argv);

    Client client;

    client.start("127.0.0.1", 12345); // 服务器IP和端口

    return a.exec();

}