“C++程序设计与训练”课程大作业
项目报告

项目名称：众包协作翻译平台

姓名：程宇笑

学号：2018010888

班级：自 84

日期：2019.7.29

目录

1 平台功能设计

互联网诞生的几十年来，网络速度不断提升、网络容量不断扩大、机器性能不断倍增。其结果，则是人类延续了几千年的，“自上而下”的树状信息交流途径，受到了一定程度的冲击，取而代之的是“点对点”的网状信息传播方式。无论是“众包平台”的兴起，还是“扁平化管理”的发展，又或者是“去设备化”的雏形，无一不体现人类社会**“内容”大于“机构”**的发展趋势。

“众包翻译平台”体现的即是一种“用户创造内容”的创新民主化，因此该平台从设计上，自底层到用户界面均贯彻**内容主导**的理念。强化“文章”（或“任务”）的存在，而弱化“模块”、“用户”、“管理员”等概念。

将平台命名为“译林”（英文名“WeTranslate”），“译”取翻译之意，“林”取众包之意，简洁大方。

1.1 总体功能描述

以文章（或“任务”，为了直观以下称为“文章”）为主体，一篇文章应该经历以下几个大的流程：发布，确定负责人，拆分，确定翻译者，翻译，子任务合并，提交。平台功能的主体部分可以简化为**图 1-1**。

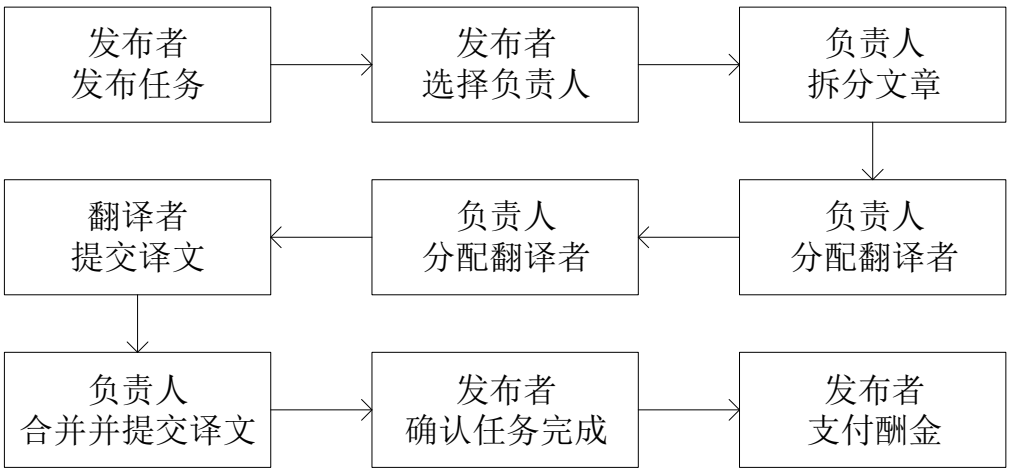

图 1-1 平台主体简化流程

上图所示的流程对应于一篇文章的许多个状态。更详细地，为每个流程对应的状态分配状态编号，以便开发与调试。**表 1-1** 列出了文章状态编号的含义，以及以文章为主体应该实现的功能。需要说明的是，状态编号的第一位有特殊含义，1 表示拆分前，2 表示拆分后，3 表示合并后，4 表示生命周期完成。

状态编号	**含义**	**对应主要功能**
0	无	发布文章，查看其他用户的文章
100	已上传，招募负责人开始	修改文章，开始招募负责人，查看并选取报名的负责人
110	已标记负责人，招募负责人结束	查看文章，修改文章，开始招募翻译者
120	开始招募译者	结束招募，查看报名的译者
130	招募译者结束，即将分配任务	拆分文章，同时创建多个状态编号为 200 的子文章
140	已拆分	无
200	子文章已创建	选择翻译者
210	已标记翻译者	查看原文，编辑译文，上传译文
215	译者开始翻译	发送反馈，查看反馈，编辑译文
220	译者正在根据负责人意见修改	发送反馈，查看反馈，编辑译文

230	子文章译文评审通过	合并子文章，同时创建状态编号为 300 的文章
240	子文章生命周期完成	无
300	子文章合并完成	提交给用户
310	负责人提交	确认翻译完成，支付酬金，拒绝接收
400	发送者已付款，款项成功分配	无

表 1-1 文章状态号及其对应含义

总体而言，每个文章对象都应经历以上流程，用户在界面进行操作的**最终结果即为更改文章状态和文章内容**。除此之外，平台还应包含一些附属功能，如登录、注册功能，查看和维护用户信息，消息盒子功能等等。

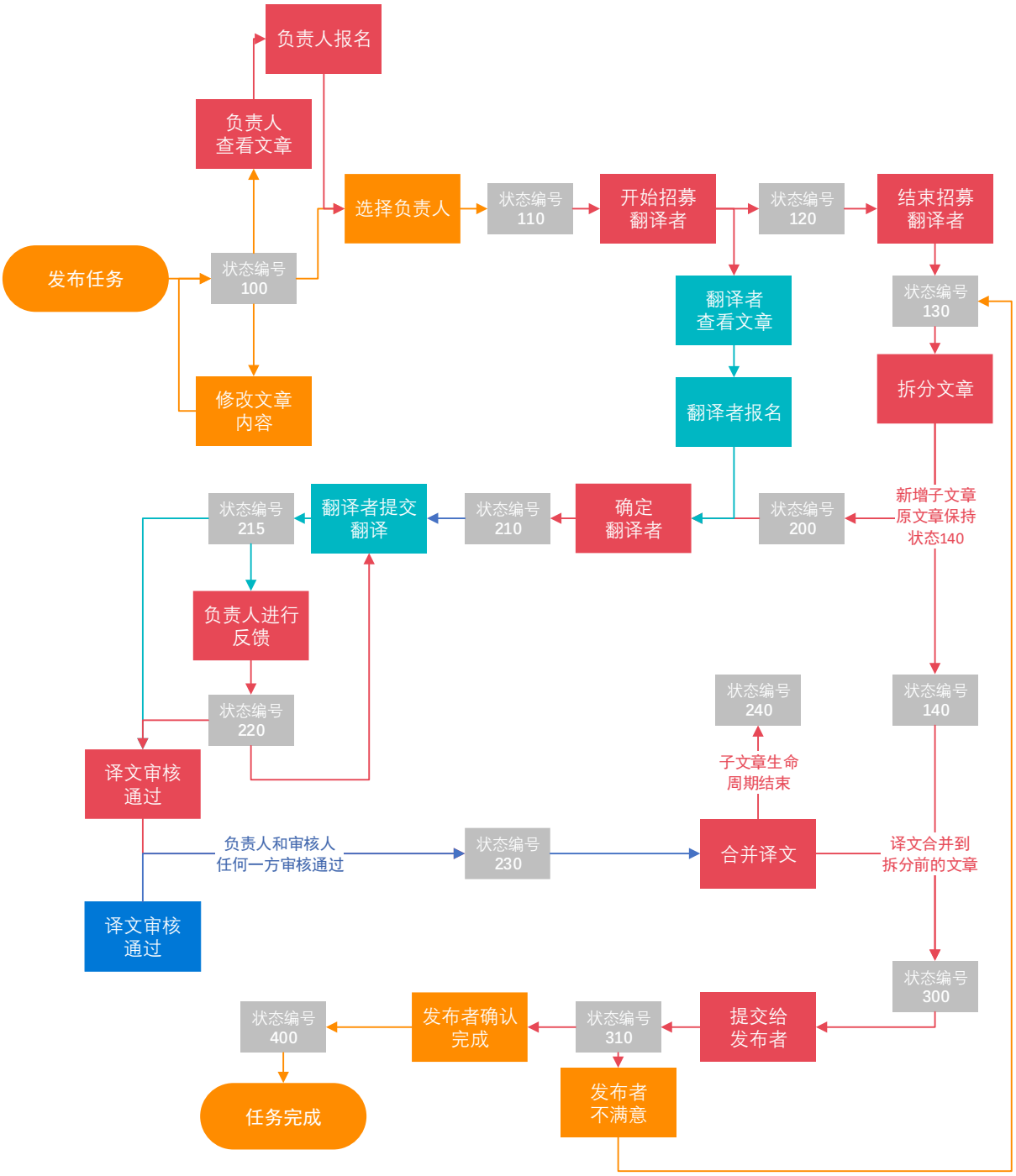

图 1-2 平台整体核心功能流程图

1.2 功能流程描述

完成文章状态号及其含义的设计之后，我们可以将平台整体核心功能的流程图绘制如图 1-2。其中灰色矩形代表文章状态改变，橙色矩形代表发布者进行的操作，红色矩形代表负责人进行的操作，浅蓝色矩形代表翻译者进行的操作，蓝色矩形代表审核人进行的操作（注意，审核人可以对任意子文章进行审核，而负责人只能对负责的子文章进行审核）。

为了实现多用户系统，需要设计注册与登录功能：一个用户可以用多种身份登录：发布者（Sender）、负责人（Regulator）、翻译者（Translator）、审核人（Supervisor），登陆后则进入该身份对应的界面。原则上讲每个用户都有进入四种界面的入口，但登录为不同的角色有不同的前提条件。注册时要求进行资质认证，为了开发和演示方便仅列出示例文字作为示意：如“TOFEL 100 分”，“英语四级”等。

登陆后用户能对文章状态进行修改，也可以进行如报名、删除、增添等操作，文章和用户、用户和用户之间的交流用“请求”（Request）机制完成，例如报名做负责人、报名做翻译者、文章状态更改通知、用户状态更改通知、负责人反馈均是一种请求。

2 平台结构设计

2.1 程序结构设计

综合考虑开发难度、界面效果、层次结构以及项目要求，**选用 Qt Quick 作为开发框架**，选择 MySQL 进行数据存储，以 **C++作为核心程序开发语言，QML 作为界面设计语言**，QML 中内嵌**少量 Javascript 代码**，进行轻量的界面逻辑处理、并与 C++交互。

Qt Quick 最初在 2010 年独立于 Qt 发布，设计理念独树一帜，使用全新的声明性脚本语言 QML 作为界面设计语言，同时支持 C++进行后台核心数据处理，还兼容 Qt 运行库，QML 和 C++可以通过信号槽机制或对象的调用进行交互。QML 本身语法形似 CSS，但支持内嵌 Javascript 语句块，这给予其极高的灵活性和可定制性。除此之外，QML 语言设计的界面本身就是动态的、响应式的，并能很好地利用 GPU 进行渲染，结合其双向绑定的属性（Property）系统，可以设计出极具设计感的界面和交互逻辑。

如此一来，QML 处理控件属性，Javascript 处理界面逻辑，C++处理后台数据操作，各司其职，秩序井然，再也不会存在 C++数据、流程、控件操作掺杂在一起的混乱状况。我在使用 Qt Quick 进行开发的过程中，虽经历非常多的困难（成体系的中文资料几乎为零），但其先进的设计理念和极富魅力的框架结构还是让我受益匪浅。

经过构思、设计、调整、实现，将该平台软件实现的程序结构从底层到顶层划分为五个层级：**Storage – Data – Model – Interaction – View**，即 **存储 – 数据对象 - 模型 - 交互 - 视图**，如**图 2-1**。五个层级均主要服务于“文章”这个基础对象，体现“内容至上”的设计理念。对五个层级功能及相互关系阐述如下。

1. **Storage（存储）层级（C++）**：用于与数据库直接交互，包含数据库的连接、数据的下载、保存、更新以及上传到数据库的功能，还包含一些存储相关的工具函数。
2. **Data（数据对象）层级（C++）**：各数据的实体保存位置，包含一些操作及读取数据的方法。对象的指针组合为 QList（或 QVector）保存在 Storage 层级。Data 可以组合为 Model 再通过 Interaction 层级与界面进行交互，也可以直接通过 Interaction 层级与界面进行交互。
3. **Model（模型）层级（C++）**：根据界面显示需求将 Data 组合为 Model，通过 Interaction 层级，驱动界面中 ListView、GridView 等控件的显示。
4. **Interaction（交互）层级（C++, Javascript）**：根据用户的需求，要求 Model 层级（或 Data 层级）向 Storage 层级查找、组织、更改所需数据，并将数据传递给 Javascript，从而驱动视图的显示。
5. **View（视图）层级（Javascript, QML）**：读取 C++和 Javascipt 传递的数据，修改 QML 对象的属性从而达成动态显示效果。响应用户操作事件，通过信号槽或直接唤起函数向 C++或 Javascript 传递数据更改或显示请求。

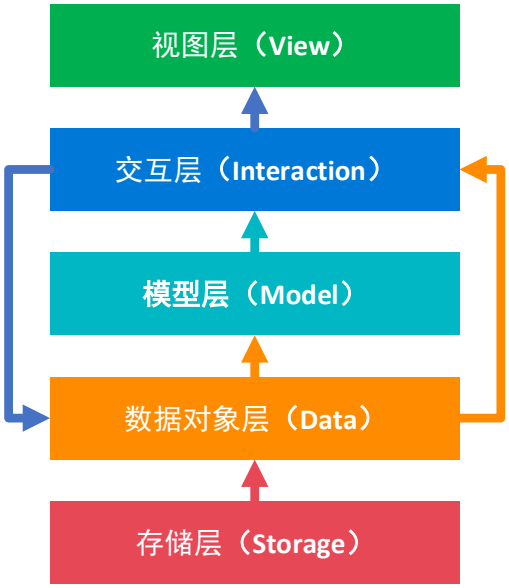

图 2-1 五层架构示意图

数据库方面，由于我曾有少量的 web 开发经验，也在开发、维护一台阿里云轻量服务器，故在该服务器上配置 MySQL 数据库，直接使用**远程连接存取云端数据**，不仅可以保证断电不损失数据，甚至可以保证更换客户端也不损失数据。

2.2 用户界面风格设计

秉承“内容大于机构”的设计理念，该平台急需一种既丰富灵活、又不喧宾夺主，既扁平简约、又不过于简单的设计规范。微软于 2017 年发布的全新设计规范 **Fluent Design** 则能很好地满足需求。Fluent Design 脱胎于 Windows 8 的 Metro UI，已经在 Windows 10 中得到初步运用。与 Google 的 Material Design 类似，Fluent Design 模仿日常生活中的常见材质，来达成自然的交互效果。**“亚克力”（Acrylic）**就是该系统中最主要的材质，在磨砂亚克力版上展现的内容，锐利和柔和完美结合，既有光影效果带来的科技感，也能很好地凸显内容本身。

本平台使用“亚克力板”作为展示内容的托盘，选取蓝色作为强调色（#0078d7），**“等线”**作为统一字体，添加边框和表面光影效果。“亚克力板”**（Acrylic Block）**也是进行交互的主要底层控件，决定了按钮、选择框、输入框等控件的显示效果。

遗憾的是，由于 Fluent Design 是一种非常新的设计理念，Qt Quick 对其控件支持并不完整，仅按钮、选择框等部分控件效果较好。得益于 QML 的灵活定制性，我从头实现了底层控件“亚克力板”**（Acrylic Block）**，并派生出许多子控件，保证了设计风格的完整实现。

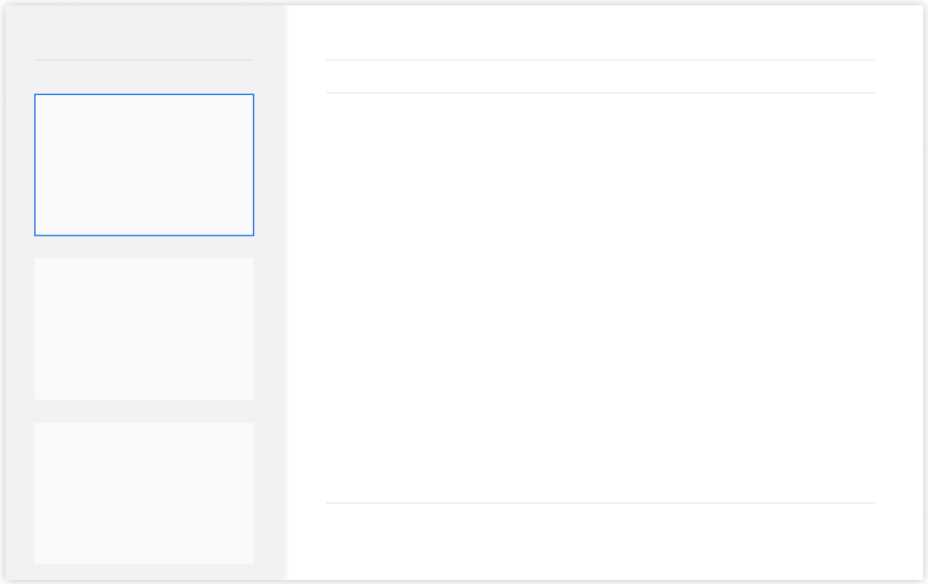

图 2-1 用户界面背景的设计稿，左栏以亚克力块的形式承载文章预览和选取功能，
右侧大块区域服务于文章内容的查看、修改等具体操作。

3 程序详细设计

以“文章”为功能主体，结合面向对象的程序设计思想，平台流程的本质就是对“文章”对象属性的读取、显示、修改、保存。“文章”属性应包含状态号（StatusCode），描述和内容（Title, Content），绑定的用户（发布者：Sender，负责人：Regulator，翻译者：Translator）。“文章”对象应有两种状态：文章（Article）状态和子文章（SubArticle）状态，并以状态号进行区分。

结合先前设计的 Storage – Data – Model – Interaction – View 架构，程序需要实现的基本模型可以这样描述：**存储层**从数据库读取数据，封装为多个**数据对象**保存在内存中，根据**交互层**的界面显示需求组合为模型，与**视图层**的 ListView/GridView 绑定。当用户在界面上进行操作，导致**视图层**的属性改变，触发**交互层**程序语句，从而通过 Index 读取**模型层**中的单个**数据对象**，并进行修改，最后**存储层**检测到数据发生改变后进行数据上传。

3.1 对于代码命名风格的说明

本项目的代码选用**驼峰命名法**而非建议的匈牙利命名法，理由陈述如下：

- Qt 以及 Qt Quick 运行库完全遵循大小驼峰命名法，如类名为 QDebug，而函数名为 qDebug()。若在项目中使用匈牙利命名法，将导致命名混乱、难以开发。
- 个人认为匈牙利命名法的前缀规则存在设计上的不合理，科学的程序开发应该是将比较机械和固定的操作交给编译器和 IDE，而把核心的思维和构造过程交给程序员。匈牙利命名法的前缀清晰地标明了作用域和数据类型，但却是把繁琐的数据类型等检查工作强加于程序员，在编译器和 IDE 不断进步的情况下，已经失去了意义。此外，前缀的字母存在随机性，与下划线组合在一起意义难以辨明，面对大量代码时眼花缭乱、影响开发效率。
- 该项目的开发仅为一人团队，故优先遵循库函数的命名风格，而非团队原有代码命名风格（因为不存在原有代码）。

3.2 类结构设计

本平台程序结构的五个层级分别包含以下一些类。需要说明的是，实际上该平台的数据库封装形式，是受推荐的两种比较合理的范式的结合。详细的 UML 类图可见图 3-1，若显示不清晰可查看附件。

1. **Storage 层级：** GlobalStorageComponents, StorageUnit
2. **Data 层级：** MyArticleObj, MyRequestObj, MyUserObj
3. **Model 层级：** ArticleList, RequestUserList, MessageList
4. **Interaction 层级：** AbstractPage, LoginPageHandler, SenderPageHandler, RegulatorPage-Handler, TranslatorPageHandler, SupervisorPageHandler, ArticleInfoPageHandler, UserInfoPageHandler
5. **View 层级：** 若干 QML 文件，内嵌少量 Javascript 代码

3.2.1 存储层包含的类

GlobalStorageComponents　数据库封装类。包含一个**后台线程**（防止阻塞前台线程造成程序运行不畅，refreshDataInNewThread() 函数）。程序启动时从远程数据库加载数据直到成功（downloadAllData），并组合为数据对象，之后每 200ms 检测**每个数据对象的更改状态**（ModifyStatus），若有新增或更改则进行上传（uploadAllData()），上传成功后消除更改状态。该类也向上层提供索取和新增数据对象的服务，并封装一些广泛使用的工具函数，例如：

- QString decodeStatusCode(int): 解析状态号对应含义。
- inline int getAnArticleId(): 自动生成一个文章 ID，比现存所有文章都大。

StorageUnit　数据对象基类，派生出 MyArticleObj, MyRequestObj, MyUserObj 等数据对象类。该基类的主要目的是标记更改状态（ModifyStatus）。包含一个枚举类型，有 Unchanged, Changed, New, Deleted 四种状态。

以上两个类共同作用，既保证了数据上传的完整性（每 200ms 都检测一次整体数据情况），也可以使得数据可以在后台实时上传；既不需要重新上传所有数据，又无需上层系统关心对数据库的操作。实际上结合了数据库封装第一范式和第二范式的优点。

3.2.2 数据对象层包含的类

MyArticleObj　文章的属性及其操作封装类，派生自 StorageUnit 类，**由于上层有封装良好且分工明确的流程类，再加上状态编号机制，故无需派生出发布者文章、负责人文章等类。**该类的属性较多，包含文章 ID，描述（标题）和内容，翻译后的标题和内容，状态编号，相关用户的 ID，状态编号，文章酬金等属性。该类的成员函数主要是读写这些属性的函数，若属性更改或本身是新增对象则在 StorageUnit 基类中标记状态。

MyRequestObj　“请求”的属性及其操作封装类，派生自 StorageUnit 类。实际上**“请求”有多种类型**（Type），“1”代表报名为负责人的请求，“2”代表报名为翻译者的请求，“3”代表负责人对文章翻译的反馈，“4”代表文章或用户状态更改的通知。该类属性包含请求 ID，请求内容，请求发送者，请求对应的文章，请求时间等。成员函数与 MyArticleObj 情况类似。

MyUserObj　用户属性及其操作封装类，派生自 StorageUnit 类。一个用户可以以多种身份登录，但用户信息只有一份。用户属性包括用户名、密码、积分、余额、认证信息等。成员函数与 MyArticleObj 情况类似。

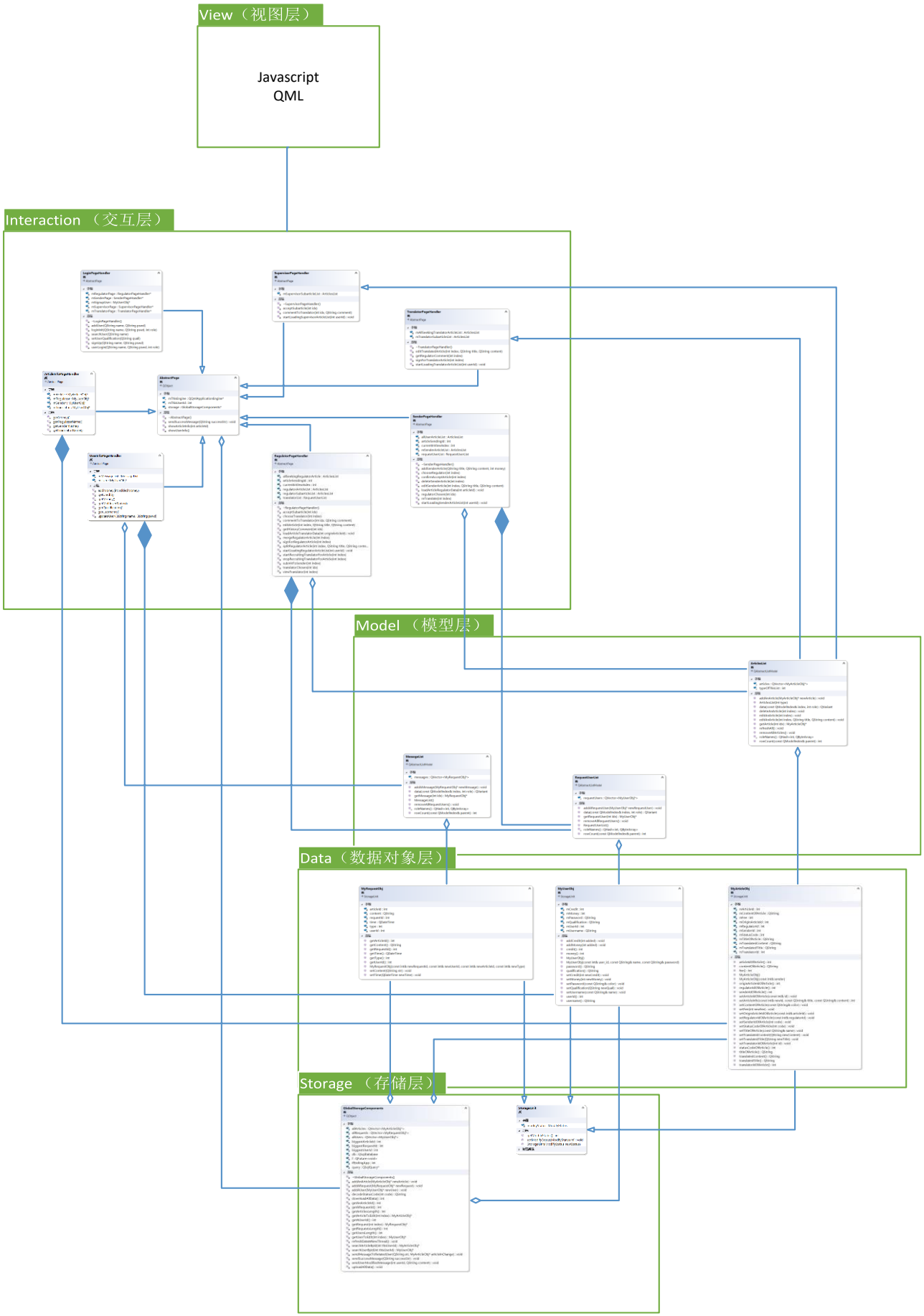

图 3-1 详细 C++程序的 UML 类图（更清晰的版本可见附件）

3.2.3 模型层包含的类

ArticleList 文章列表类，用于视图层中 ListView 和 GridView 的显示，继承自 Qt 的 QAbstractListModel 类，重载父类的若干方法，对一些成员函数和变量说明如下：

- int *rowCount*(QModelIndex): 要求重载的方法，用于计算总的行数。
- QVariant data(QModelIndex, int): 要求重载的方法，用于查询应该显示的数据。
- QHash<int, QByteArray> roleNames(): 注册到 QML 的变量名的表，QML 界面根据这些变量名查询数据进行显示。
- QVector<MyArticleObj *> articles: 存储所有文章的数据结构，由于需要保证文章对象实体只有一份，使用指针

RequestUserList: 显示报名的用户列表。结构与 ArticleList 类似。

MessageList: 显示消息盒子中的消息列表。结构与 ArticleList 类似。

3.2.4 交互层包含的类

由于该层级与 QML 视图层级直接交互，故包含大量的 Q_INVOKABLE 函数（可以在 QML 文件中直接调用）和信号（与 QML 连接）。

AbstractPage 各用户后台流程类的基类，用于实现一些页面共有的方法和信号，以及规定派生流程类需要实现的接口。对关键的成员函数和变量说明如下：

- static GlobalStorageComponents* storage: 静态成员 Storage，实例化存储层类 GlobalStorageComponents,这是交互层需要查询、更改、新增、删除数据必须使用的工具变量。
- Q_INVOKABLE void showUserInfo(): 启动展示当前用户信息的页面
- Q_INVOKABLE void showArticleInfo(int): 启动显示传入用户 ID 对应用户信息的页面。
- void sendErrorMessage(QString): 信号，通知 QML 显示错误信息。
- void sendSuccessMessage(QString): 信号，通知 QML 显示操作成功信息。
- virtual void startPage(QQmlApplicationEngine *engine)=0: 纯虚函数，**派生类必须重写，在启动主页面时必须调用**，重写的函数内容为渲染 QML 页面以及注册 C++对象到 QML 中。

LoginPageHandler 登录、注册页面的后台处理类，派生自 AbstractPage。主要的成员函数有

- Q_INVOKABLE void loginInit(QString, QString, int): 根据 QML 传来的用户名、密码、登录角色进行验证，验证通过则启动对应的界面。
- Q_INVOKABLE void signUp(QString name, QString pswd): 根据 QML 传来的用户名、密码进行注册。设置认证信息后，先调用 searchUser(…)函数验证是否已存在该用户，若没有则调用 addUser(…)增加用户，注册成功。

SenderPageHandler 发布者页面流程处理类，派生自 AbstractPage，定义大量的流程逻辑函数，详细的信息可以查看代码及注释得知，在这里说明几个关键的成员函数和变量：

- void startLoadingSenderArticleList(int): 根据界面显示需求，从 storage（存储层）查询所需数据对象，组合为新的 ArticleList 类的对象（模型层），用于 GridView 的显示。
- void loadArticleRegulatorData(int): 根据当前正在查看的文章，加载报名的负责人列表（其模型是一个 RequestUserList 类的对象），在新的选择负责人子页面中显示。

RegulatorPageHandler 负责人页面流程处理类，派生自 AbstractPage。

TranslatorPageHandler 翻译者页面流程处理类，派生自 AbstractPage。

SupervisorPageHandler 审核人页面流程处理类，派生自 AbstractPage。

UserInfoPageHandler 用户信息页面的流程处理类，派生自 AbstractPage。QML 可以调用其中的函数获取用户各种信息，以及修改用户名和密码，显示消息盒子消息列表（其模型是一个 MessageList 的对象）。

ArticleInfoPageHandler 文章信息页面的流程处理类，派生自 AbstractPage。QML 可以调用其中的函数获取文章的各种信息。

QML 中内嵌的 Javascript 代码　以上每个流程处理类均有对应的界面（基类除外），界面布局由 QML 描述，其中内嵌的 Javascript 语句用于和 C++交互、处理轻量界面逻辑，更多的 QML 流程逻辑详见 3.2 节。

以发布者页面为例，登陆成功时 SenderPageHandler 的一个对象（senderPageHandler）被创建，之后调用该类的 startPage()函数将该实例、文章列表的 Model 注册到 QML，然后启动 SenderPage.qml 文件。该 QML 文件可以调用 senderPageHandler 对象中注册为 Q_INVOKABLE 的函数获取信息，根据文章信息由 SenderEditor.qml 文件中的 Javascript 函数 refreshEdit(…)为控件分配不同的显示信息以及操作动作。

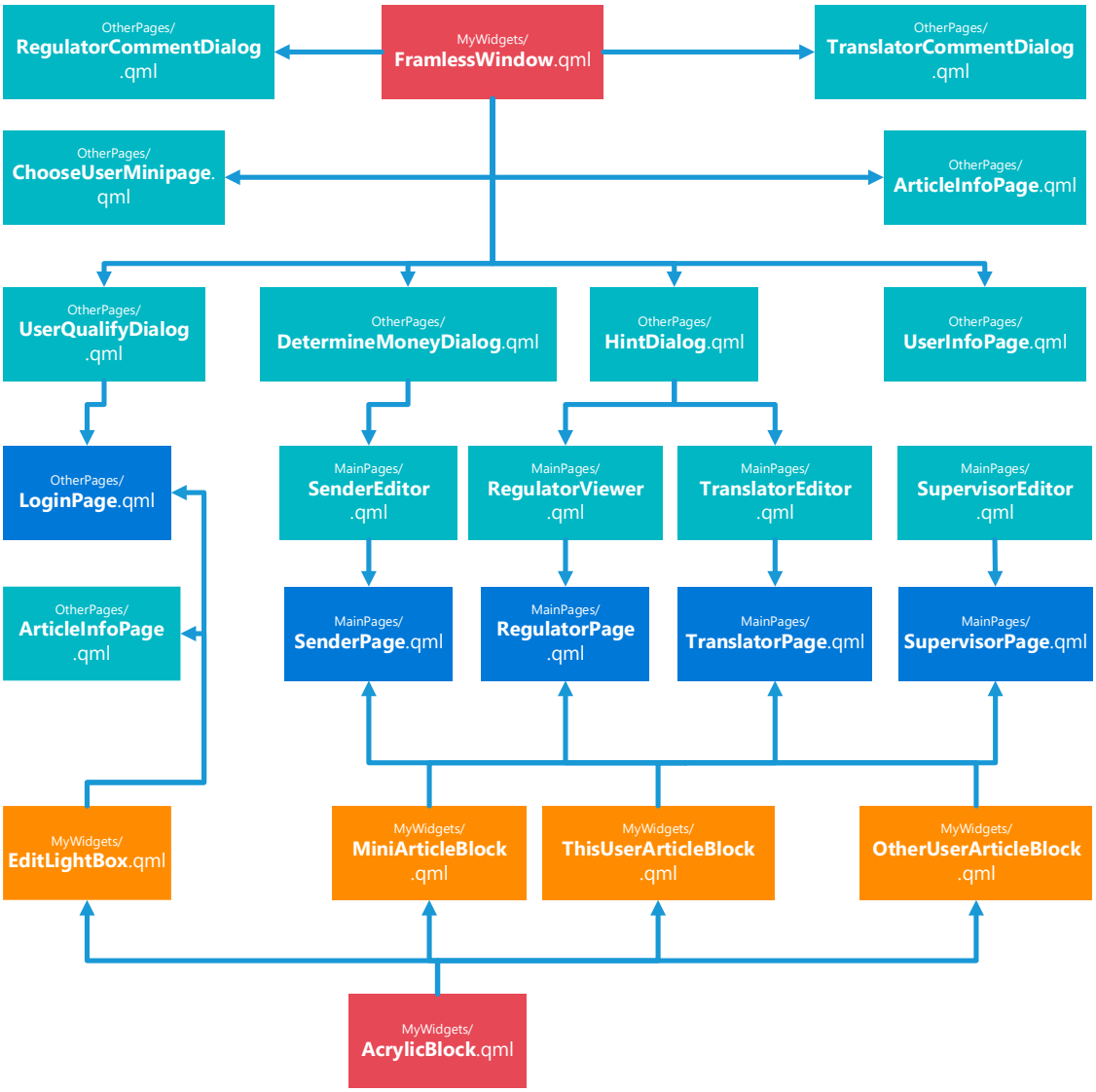

图 3-2 QML 文件调用关系

3.3 视图层结构设计

该项目包含大量的 QML 文件，由于我从头实现了某些符合 Fluent Design 设计规范的控件，他们各自有一些调用关系，关键调用关系如图 3-2，其中蓝色代表顶层主页面，浅蓝色代表上层控件，橙色代表中层控件或子页面，红色代表底层控件。

由图 2-1 可知，顶层主页面分为左右两部分。左侧是一块亚克力背景板，上面放置若干中层控件 ArticleBlock，用于显示多篇文章预览信息。ArticleBlock 调用了底层控件 AcrylicBlock（亚克力块），

以达成对应光影和交互效果。右侧是白色底板，整体包装为一个上层控件 Editor（如 SenderEditor，RegulatorViewer 等），用于文章的编辑和显示。

移除原生 Windows 有碍观瞻的窗口边框后，再实现拖动和阴影等效果，就得到了底层控件无边框窗口 FramelessWindow，各种提示框、警告框、选择认证信息、显示用户或文章信息的子页面均使用无边框窗口控件，并尽量简化结构。

更详细的信息需要查看代码及其注释，更多有关界面设计的效果见第 4 节。

3.4 数据库/文件结构设计

本项目使用远程 MySQL 数据库，建立了三张表（TABLE），分别为：

- users：存储用户信息

user_id	用户 ID（唯一）	**quali**	认证信息（字符串）
user_name	用户名（唯一）	**credit**	积分
password	密码	**money**	余额
create_time	创建时间		

- articles：存储文章信息

article_id	文章 ID（唯一）	**origin**	子文章拆分前的原文章 id
title	标题（描述）	**t_title**	翻译后的标题
content	内容	**fee**	酬金
sender	发布者 ID	**translator**	翻译者 ID
regulator	负责人 ID	**t_content**	翻译后的内容
create_time	创建时间	**curr_status**	当前状态编号

- requests：存储“请求”信息，请求有四种类型，每种类型情况不同。
  1. 报名为负责人 **(type=1)**：user_id 为报名的负责人 ID，article_id 为目标文章 ID,content 为空。
  2. 报名为翻译者 **(type=2)**：user_id 为报名的翻译者 ID，article_id 为目标文章,content 为空。
  3. 负责人向翻译者反馈的信息 **(type=3)**：user_id 为负责人的用户 ID，article_id 为反馈的子文章 ID，content 为反馈内容。
  4. 状态更新通知 **(type=4)**：user_id 为接收通知的用户 ID，article_id 无效，content 为通知内容。

request_id	请求 ID（唯一）	**user_id**	相关用户的 ID
article_id	相关文章的 ID	**time**	发送时间
content	内容	**type**	类型
create_time	创建时间		

4 界面详细设计

简洁美观的界面是本平台最大的亮点之一，使其成为可能的是整个用户界面系统秉承的**“内容大于结构”**的核心理念。为了达成这样的效果，本平台从头实现了符合微软 Fluent Design 设计规范的，仿亚克力磨砂质感的若干控件（Acrylic Block）。

本平台在设计界面时，先使用设计软件 Figma 进行精细化设计，再将界面实现到程序中，这样更能明晰需求、高效开发。全部设计稿可见附件。

4.1 控件

Acrylic Block　使用多层遮罩实现亚克力块效果的底层控件，最顶层是**边框层（Border）**，模拟光

线照射在亚克力表面的折射和全反射效果，对应于光标悬浮、点击、选中、选中后悬浮等多种状态的效果不同；之后是**追光层（LightChaser）**，模拟光线照射在磨砂材质表面的漫反射效果，追光位置跟随悬浮的光标；其次是**内容层（InnerComponent）**，可以给上层控件调用以添加内容；再其次是**底板层（BackGround）**，模拟亚克力磨砂材质的颜色；最后是**外围层（Outside）**，是承载亚克力板的背景颜色。这种设计既符合直觉，又不过于繁杂，让人在自然的交互氛围中将全部精力集中于内容本身。

当用户对亚克力块进行点击，内部内容则相应发生位置或大小的改变，以模拟物理**受力形变**。此外还为亚克力块增加了拖拽、选中等定制功能。

Article Block 调用上述亚克力材质底层控件，达成美轮美奂的光影效果。增加表示状态的图标（每种状态对应一个图标，简洁直观），文章的描述和内容的预览的显示。效果如图 4-2。追光的调教尤为重要，不能过浓从而遮盖内容，又要产生新鲜感从而刺激用户对内容的兴趣。

图 4-2 本项目实现的 Article Block 设计稿及其对应的几种光影效果（选中，无效果，悬浮）

Edit Block 调用上述亚克力材质底层控件，去除追光效果，加粗边框，增加 Text Edit 控件，做了大量调整以达成符合 Fluent Design 设计规范的输入框效果，效果如图 4-3。

Frameless Window 某些情况，特别是要显示较小的弹出窗口时，Windows 原生控件的标题栏显得不够简洁和“内容突出”。去除标题栏，重新实现窗口拖动功能，效果如图 4-4。

图 4-3 Edit Box 控件设计稿

图 4-4 无边框窗口控件实现的对话框设计稿

有了这些控件的支持，我们终于可以开始搭建界面了。

4.2 登录界面与主界面

4.2.1 登录界面

登录页面是平台的总入口。为避免引起混淆和逻辑不清，用户在登陆时必须选择登录角色，登录后只能进行该角色对应的操作。（注：忘记密码功能在该平台中并不需要，也就并未开发。）

登录页面与其他页面相同，为居左重心，将强调色施加于左下方，更显稳定。左上角可见应用图标，能清晰明确地辨明“翻译”含义，同时增加了一个视觉重心。点击注册，会检查该用户名是否已注册，若没有则会弹出对话框提示选择用户认证信息（仅作为演示）。

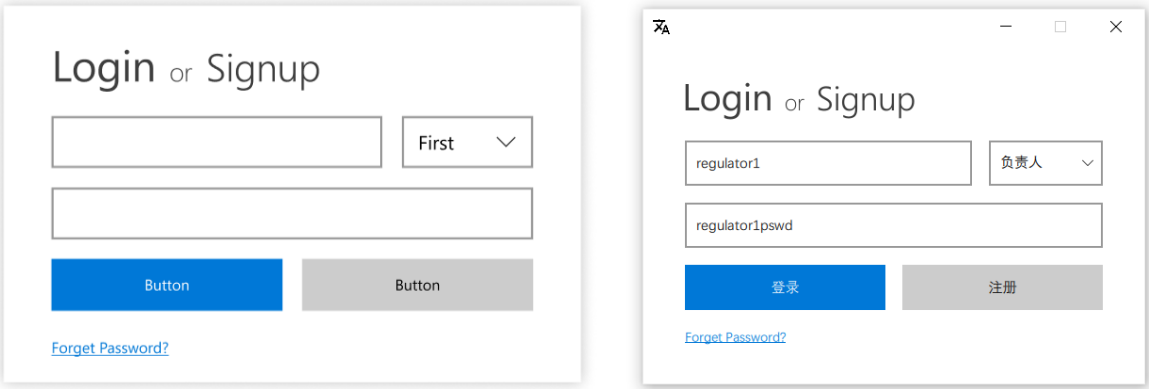

图 4-5 登陆页面设计稿及实际效果

4.2.2 主页面

主页面分为两个部分，左端为**托盘展示区**，是视觉的首先落脚点，在浅灰色背景上叠放亚克力块（Acrylic Block），可以上下滚动显示更多文章，并点击选中某篇文章。当用户上下滑动、光标移动时，追光效果跟随光标、亚克力块的光线全反射效果随叫随到，给人自然的交互体验，为本是繁琐的查找、检索过程增添了活泼与调皮。发布者页面的左上角为**新增文章**按钮，放在最为突出的位置，这也是发布者页面的功能重点、一切内容的源头。

右侧大块区域服务于对选中文章的操作。左下角为**更改操作区**，左侧按钮为**主要动作**（例如上传、修改、开始招募）、右侧按钮为**次要动作**（例如反馈、查看报名状态），左侧按钮增加强调色以设置视觉重心。右下角为**查看操作区**，在负责人、翻译者、审核人页面可以切换原文与译文的查看。

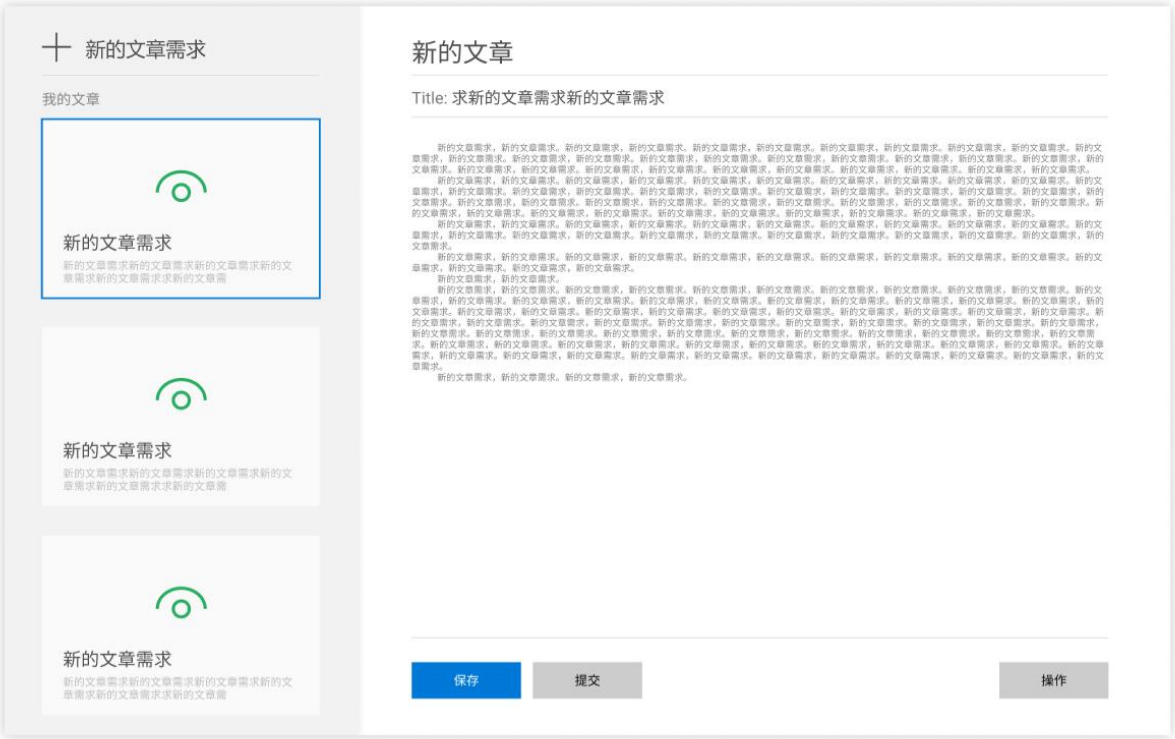

图 5-6 发布者主页面设计稿

在实际实现中，为托盘展示区的每个文章的不同状态，分配简洁明了的状态标识，如图 4-7。文章操作区右上角增加两个**下拉菜单启动按钮**，点击可以启动**文章详情**窗口（显示文章酬金和相关用户）和

用户信息窗口（显示和维护用户信息，查看**消息盒子**）（如图 4-8，4-9），分别是两个悬浮的无边框窗口。

如果文章较多，托盘区仅一列文章上下滑动略显繁杂，可以通过“展开”托盘将一列变为三列，形成一个“模态窗”，极具动态感，如图 4-8。

图 4-7 状态编号对应的状态图标

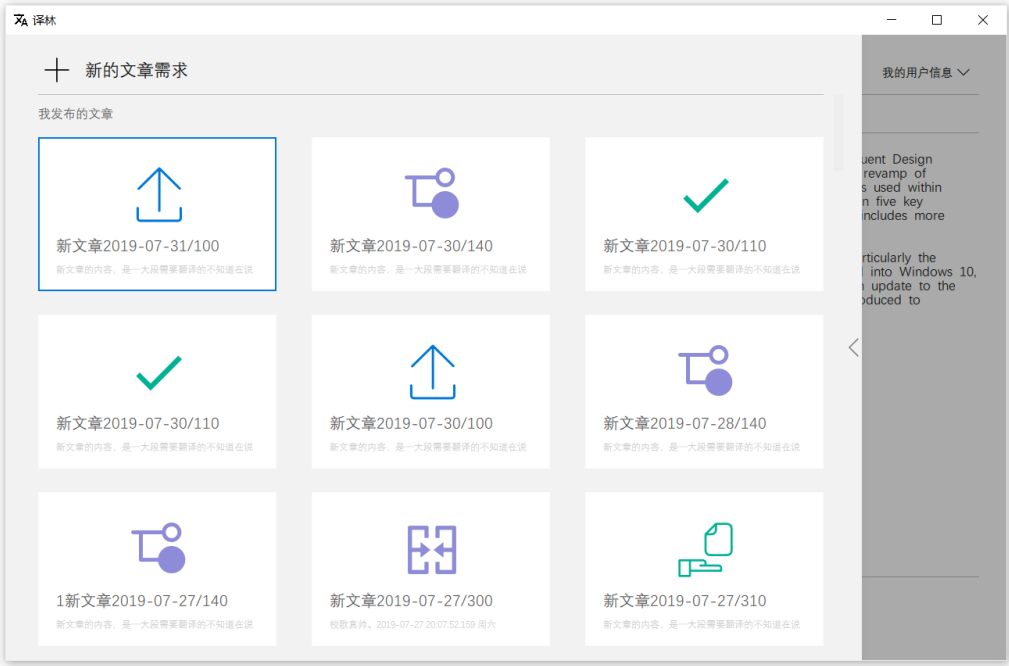

图 4-8 发布者页面实际效果截图

图 4-9 用户信息页面和文章详情悬浮窗

负责人和翻译者页面效果类似，但去除了发布文章按钮，增加了代表子文章的小亚克力块，效果如图 4-10，主次分明、一目了然。

图 4-10 负责人页面

为了增加操作的直观性，对于“亚克力块”这样扁平效果的拟物概念，为其增添“拿起”和“拖拽”动作，在左侧托盘区可以用鼠标或触摸手势选中拖拽亚克力块，并拖放到右侧的操作按钮处，完成主要操作（支持删除文章、分配翻译者的操作）。

视觉效果方面，为了凸显出“拿起”的视觉层次感，增加边界处阴影。为了达成“可用性”提示，在拖放到位后对操作按钮进行背景色改变处理，如图 4-11。

图 4-11 拖拽效果设计稿

4.2.3 对话框

本项目使用无边框窗口（Frameless Window）达成 Fluent Design 风格的对话框效果，失去焦点即自动隐藏，效果如图 4-12。

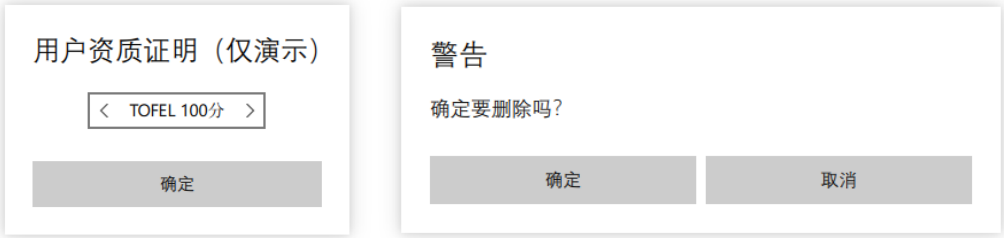

图 4-12 对话框实际效果截图

4.2.4 提示栏

在 C++程序中，可以使用全局信号 sendErrorMessage(QString)或 sendSuccessMessage(QString)通知 QML 界面显示提示消息，浅绿色和红色分别代表操作成功和操作错误的提示。提示栏本身也是亚克力块风格，从顶端缓缓滑下，短暂停留后又缓缓上滑，好像是在捉迷藏。

图 4-13 提示栏截图

4.2.5 动画效果

前文说过，使用 QML 设计的界面本身就是动态和响应式的，这就给予了动画效果极高的可能性。该平台的界面在提示栏、列表视图、拖拽操作、文章托盘展开、弹出框等多处使用动画效果，实际效果十分惊艳，还需在使用中体会。

5 项目关键点与亮点

5.1 附加功能实现

5.1.1 审核人角色

本平台在发布者、负责人、翻译者角色之外增加了审核人角色，系统将审核人设计为较高权限的角色，可以查看任何一篇正在翻译、等待审核的子文章，并审核通过。审核人和负责人任何一方审核通过，则将该子文章标记为“审核通过”状态。

5.1.2 拖拽快捷操作

在鲜明的“内容大于结构”的设计理念下，文章本身的概念被提到了突出位置。是用户操作文章，而不是文章附属于用户。将一篇“文章”概念抽象为一个亚克力块，既然亚克力块存在光的折射、全反射、漫反射、受力形变的效果，也就应当存在拿起、放下、丢弃、收集的操作。

该平台为拖拽手势设置了两种快捷操作：**删除任务**、**分配任务**。在发布者页面，可按住亚克力块拖动，右侧删除图标自动出现。将亚克力块拖动到删除图标，则弹出警告提示框询问是否确认删除。在负责人页面，可将子文章拖动到右侧分配按钮处，一键分配译者。

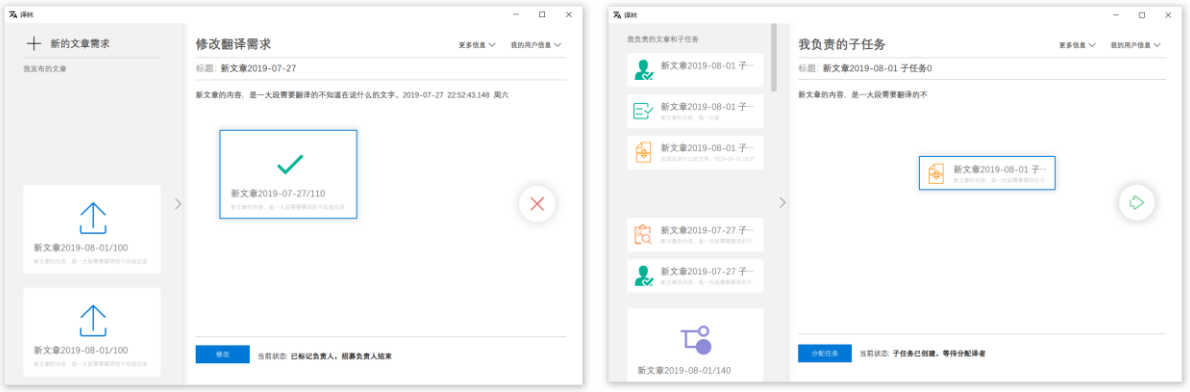

图 6-1 拖拽操作实际效果截图

5.1.3 云端上传、断网重连

既然 MySQL 本身是服务器型数据库，若仅仅将其架设在本机则完全失去了该数据库的意义。我在 Web 开发领域有一些兴趣以及少量的经验，自然而然地想到将其部署在自己一直在维护的一台阿里云轻量服务器上，繁杂的部署过程之后发现功能易用、状况良好。不仅可以保证断电不损失数据，甚至可以保证更换客户端也不损失数据。

考虑到网络连接可能出现的不稳定，加入**断网重连机制**。数据在网络状况不佳的情况会**反复上传直到确认上传成功**，如果网络断开、数据会暂停上传，**网络连接恢复后能自动继续上传**未上传的内容。

5.1.4 删除、修改文章和拒绝接收译文

为了提升用户体验，发布者在未开始招募负责人时，可以删除或修改文章，删除方式未拖拽到右侧删除按钮。负责人在合并译文提交给发布者后，发布者可以拒绝接收，修改状态号为 130（招募译者结束），重新分配任务进行翻译。

5.1.5 自动拆分

考虑到手动拆分可能产生的不准确和繁杂性，提供自动拆分功能。程序自动在空行处（而非提行）拆分，可以在拆分前调整空行位置，既具灵活性、又提升用户体验。

5.2 设计亮点

5.2.1 面向对象、层层调用

精妙的五层构型，是本项目的独特核心设计。它能很好地体现面向对象的设计思维，即“紧内聚，松耦合”。每一层次中的每一个类都**各司其职，互不干扰**；拥有共同成员或接口的相似类，被完整包纳在相应的基类中，**泛化和抽象的概念**也得到了体现。平台程序中，视图层绝不会横跨多个层级，试图操作数据库；操作界面控件的代码，与操作数据库的代码也绝不会混杂。“该是谁做的事，就拿给谁做”，如此而来，开发和调试变得有条不紊。

即便在 QML 实现的界面设计领域，控件之间也是层层封装、有序调用。不滥用信号槽，让函数的调用有迹可循。

5.2.2 容错性与可移植性

该平台最大的特点是状态复杂、流程较长。为了避免用户非常操作可能导致的程序崩溃、异常运行等问题，本项目多处设置了检查机制和错误处理手段**（注：为了演示方便并未对用户名和密码长度进行检查）**。

例如在数据库的封装类中，考虑到与云端数据库的网络连接可能存在的不稳定性，数据会在检查到上传成功后再清除标志位；如果检测到网络断开连接，数据库会尝试重新连接，网络恢复之后，之前未上传完成的数据仍然可以正常上传；如果网络状况极差，数据无法正常下载，平台也会给出相应提示，并极力保证用户的操作体验。

Qt Quick 自身拥有极强的可移植性，如果愿意，甚至可以在 Android 和 IOS 设备上运行。此外，该项目设计的界面布局几乎完全是响应式布局，可根据窗口大小对显示进行调整（但由于时间和设备的原因，仅在 1920*1080 的 PC 上进行过测试）。

5.2.3 引领潮流的平面设计风格

在我的理解中，Fluent Design 是一种典型的“扁平风格、拟物概念”的设计理念，本项目中不计成本地实现了该设计风格，不仅远离了 Windows 原生控件这一“上个世纪的审美”，更改进了单纯的“极简”和“扁平”，而是将更多的物理现实引入平面设计，其光影效果、动画效果给人极强的愉悦感。

我作为软件开发的初学者，又是自己实现了一些控件，许多设计上的把控难免有经验不足，另外还有许多设计点难以实现。比如窗口半透明高斯模糊效果，我实在是难以想到性能满足要求的解决方案。但无论如何，很明显的是，我在该平台的平面设计上倾注了心血，最终效果也非常精美。

5.2.4 多线程编程

多线程编程，自古以来就不是一个容易攻克的领域，尤其是在掺杂数据库功能之后，尤其是使用 C++ 语言。尽管如此，我在经历长时间的学习、开发、调试后，仍然将多线程技术运用到了本平台中。由于数据量大、网络延时不稳定，数据的上传是较为耗时的过程，将其独立为一个后台线程，实时刷新、检测然后上传更改后的数据，并对数据进行保护以避免可能发生的崩溃。后台线程与前台线程相互独立，但仍可以通过信号槽机制发送错误和成功信息。

6 项目总结

我们在奋不顾身地追求一件事情时，不论是在完成一个程序、还是在追求一个异性、还是在撰写一篇文章，大多看中的不是这件事情本身，而是在做这件事情时展现出来的、自己身上的品质。因为这种品质让自己感动，让自己感觉到生活的意义感，也因此深陷于此、难以自拔。对于程序设计而言，正是因为现实的不如意，我们才如此沉迷于自己能做到的这一小部分事。

我在开发之初就像将这个平台做成精品，不仅在程序结构上、也在视觉效果、交互流程上。当你一步一步达成这些效果时，一种满足感让整个人得到了升华。这是一种在日常生活中很难感受到的快乐。

跨越千险，排除万难　整个项目的开发历程超过 30 天，从 7 月 1 日开始，我就在思考整个平台的流程结构、框架选择。最终选择了相对冷门的开发框架 Qt Quick，学习 Qt Quick 的初期是极其痛苦的一个过程，中文的成体系的资料几乎为零、英文资料也几乎只存在官方文档和零星的论坛问答，但经过这段时间后，Qt Quick 两端分离的先进结构、QML 双向绑定机制体现出的极强魅力又让人难以自拔。

MySQL 驱动的配置和连接并不像听起来那么简单。但就是为了达到“云端保存”这样一个简单的效果，我先是花了整整半天在云服务器上配置好 MySQL（最大的波折是我不知道阿里云为云服务器额外设置了一层防火墙），又花了将近一天将 Qt 和 MySQL 驱动连接起来（这件事相信每个用了 MySQL 的同学都会花大量篇幅吐槽，就不再赘述）。

专注细节，精益求精　界面设计，是本平台的重要组成部分。我从来不认同工科学生只需要专注于技术，能用控制台程序把算法搞定就万事大吉。从选用 Fluent Design 设计规范开始，我就注定走上了一

条不归路，因为在技术圈子里可能并不会有很多人会在乎这种执着。

Qt Quick 对 Fluent Design 支持并不完整，那我就用最简单的矩形、最简单的画布搭建一个亚克力块控件，层层调用，总能达到效果（最值得吐槽的地方是，Qt Quick 居然连一个完整的可滚动文本编辑框控件都没有）。为了使“追光”效果尽量完美（微软官方叫做 Hover Light），我甚至花了三个小时调整渐变色的渐变参数，“圆台”型渐变比“锥”型好（所以内外径比例是 0.4 好还是 0.35 好呢），但要达到水波涟漪的效果，似乎需要非线性函数……这个动画 150 毫秒有点拖泥带水，80 毫秒又显得不合逻辑，100 毫秒似乎比较干净利落……这样的强迫症细节控的调整，远远不止这里两处。

系统规划，价值引领　本平台核心程序采用的五级架构绝非凭空得来，而是在经历大量的思索与尝试之后得到。起初我使用的其实是数据在后台实时上传的第二范式、甚至还尝试用多线程保证流畅性。后来听闻大作业要求，就花大功夫改为了整体上传，并设计了具有一定美感的五级架构。然后居然课上又讲我原来的那种方法得分最高，我气了个半死。但是功夫并没有白费，最后的实现方案实际上结合了两种范式的优点。

整个平台实际上围绕着一个核心理念“内容大于结构”，不论是界面设计还是程序架构设计，围绕着这个理念展开，就显得具有科学性、游刃有余，实际的实现过程中也有了底气。

其实人们说得没错，要是生活像写程序一样简单就好了。生活中出现的 Bug 几乎从来不给你机会调试，我们遇到的棘手难题也从来不提倡“试一下呗，不行就注释掉”，我们做过的悔恨往事也从来不支持“Git 版本回滚”，不知道该怎么办了也从来不能“查一下官方文档”……很少有什么事能像编程这样纯粹了，它不看人的颜值、不看经济实力、很少靠运气，似乎这日益膨胀的互联网泡沫就是要给现实营造一个魔幻的镜像。但是对于我而言，能有这么纯粹的爱好，能有这么纯真的快乐，就很好了，不是么？

7 附录

7.1 应用程序部署说明

Qt 虽然提供了非常强大的运行库，但对应用程序在 Windows 下部署的支持十分不友好。Qt 自带的部署工具 windeployqt.exe 可以用于查找并补全动态链接库，但并不能找到所有 dll 文件，甚至在设置插件（MySQL 驱动）目录时还存在 BUG。几经周折，耗费大量的时间，我终于写了一个自动部署 Qt Quick 应用程序的批处理命令“deploy_app.bat”，将其中的 windeployqt.exe 修改成所需目录即可。

得到应用程序及其链接库后，可以使用软件 Enigma Virtual Box 将其打包为单个.exe 应用程序，这样就可以在其他计算机上完美运行了！

7.3 文件目录说明

```
|   111README 文件目录说明.txt
|   WeTranslate_绿色版.exe                可以直接双击运行的绿色版程序，启动稍慢
|   暑期大作业-项目报告(2019)-程宇笑.docx      项目报告
|   译林 WeTranslate_Deployed.zip          部署后的应用程序压缩包，可以双击其中的 EXE 文件运行，启动较快
├──WeTranslate                            项目源文件目录
|   ├──Backups                            备份的一些其他文件
|   |       article-status-code.txt           设计的状态编号及其含义
|   |       ClassDiagram.cd                       VS 生成的类图
|   |       deploy_app.bat                    部署应用的批处理文件
|   |       Deploy_app_boxed.evb                  用于生成绿色版程序的 Enigma Virtual Box 工程文件
|   |       mysql-cmds.txt                    常用的数据库命令
|   |       mysql-columns.txt                     数据库结构
|   |       WeTranslate.ico                       应用图标
|   ├──CPP_Data                           数据对象层 C++源文件
|   ├──CPP_Interaction                    交互层 C++源文件
|   ├──CPP_Model                          模型层 C++源文件
|   ├──CPP_Storage                        存储层 C++源文件
|   ├──Deploy_addons                      部署所需的一些需要手动添加的库文件（使用批处理命令自动添加）
|   ├──QML                                QML 源文件
|   |   ├──MainPages                            主界面及其上层控件源文件
|   |   ├──MyWidgets                            中层和底层控件源文件
|   |   └──OtherPages                           其他页面源文件
|   └──Resources                          资源文件（图片和常量字符串）
|       └──status-code                          状态编号对应的图标
└──附件                                   附加的文档和文件，高清版流程图和 UML 图
```