# “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：郑彭畅

学号：2018011549

班级：自 85


日期：2019.8.1

# 目　录

# 1 平台功能设计

这是进行复杂软件开发的第一步，即需求分析。此部分需要根据自己对大作业的理解，详细描述系统将要实现的功能（此部分若直接抄袭大作业说明，项目报告部分的分数直接得 0 分）。

①项目背景

众包，百度百科定义为“众包指的是一个公司或机构把过去由员工执行的工作任务，以自由自愿的形式外包给非特定的（而且通常是大型的）大众志愿者的做法。”[1]也即是说，众包协作平台中的任务，对于平台中的任意用户而言，只要符合条件，则可以以任意角色参与任务。例如在本次大作业众包协作翻译平台中，用户可以作为**发布者**发布一个新任务，同时也能够以**负责人**、**翻译者**的身份参与任务。而用户与角色的交互，则受到用户在任务中的角色不同的影响，对于同一个任务，用户相应的能够完成不同的操作。

②.基本需求分析

笔者认为，在众包平台的背景下，所设计的平台功能至少应满足以下基本需求：

②.1 用户与任务的交互

用户与任务的交互，主要是由用户对任务所进行的单向操作，而用户相应的任务（例如用户所发布的、负责的、参与翻译的）则应该对用户的相关操作有区别地进行反馈。

以下内容首先由用户对任务可进行的主要操作展开，包括用户查看任务，用户参与任务（发布、报名、负责、翻译）等相关内容予以展开。

②.1.1 用户查看任务

首先应该注意到，平台中的用户在查看任务时可能有用户以下几种不同状态：

- 作为任务发布者
- 作为任务负责人
- 作为任务译者
- 作为任务负责人且待审核通过
- 作为任务翻译者且待审核通过
- 作为与任务无直接联系者

另一方面，用户所查看的任务本身也有以下几种不同状态

- 正在招募负责人
- 正在招募翻译者
- 正在翻译
- 翻译结束

分析后不难得到：作为任务发布者，其对于任务具有最大的掌控权，理论上能够查看到该任务的所有相关信息，并且具有对任务的部分参数直接修改的权利。在设计时，发布者所查看的界面展示的信息是相对最全面的，同时发布者用户在查看任务时，界面要提供足够的接口保证发布者对于任务信息的

[1] 引自百度百科。

查询与更改。此部分将在后文 1.1 总体功能描述中进一步阐明。同时，任务发布者对任务的查看，随着任务状态的改变，所受到的影响相对而言是最小的，例如，当任务由正在招募负责人变成正在招募翻译者后，对于发布者，进行查看任务时比之前一状态，将会看到确定好的译者名单以及任务状态的改变，而其他信息并不会受到影响。

作为任务负责人，其对于任务的操作受限于任务发布者，但当他成为负责人，基本上就具有对实际任务进行流程的控制权力。在查看任务时，负责人能够查看到任务进行的基本信息，同时具有一定的改变任务状态的权力（笔者分析，应该是从“正在招募译者”转变为“正在翻译中”这一状态的改变）。

作为任务翻译者，当由负责人对原始任务进行分割后，译者获取译者所负责的任务（以下称之为子任务）。对于译者的子任务，译者应该具有对于任务原文内容、任务目标语言、任务周期的基本浏览权限，同时，为了实现译者互校功能，译者应该也能够查看同一原始任务下其他译者的任务内容，但仅仅拥有访问的权力而不具有编辑、修改的权力。

作为任务负责人报名人、任务译者报名人以及无关用户，其对于任务的查看权限依赖于任务的状态。但总体而言，查看任务时都应该实现对任务描述、任务发布者、任务报酬、任务报名截止时间、任务周期等影响用户选择任务等信息的呈现。

②.1.2 用户操作任务

用户参与一个任务，由平台的基本描述可以知道，应该具有以下三种身份：

- 作为任务发布者
- 作为任务负责人
- 作为任务译者

任务发布者对于任务的基本操作，包括编辑任务内容、发布任务、查看任务进度、确认任务负责人、结束一个任务。在作业设计要求中，并不要求发布者在任务翻译进行过程中实现对任务的检查，笔者分析在任务翻译任务中，负责人应该是全权负责任务的进行，发布者只对负责人所提交的最终进行审核，故也并没有在中间另外设计接口。需要注意的是，在本任务的模板平台**译言古登堡计划**中，发布者对于所报名的负责人，具有一定的筛选方式以及绝对的选择权利，如发布者可以通过试译稿来确定负责人报名者是否合格，也能够在没有合适的负责人时适当延长报名截止时间。这些完善的功能笔者认为是对本平台的很好补充。

任务负责人对于任务的基本操作，包括设定翻译者报名截止时间、确定翻译者、分割任务、查看子任务、对任务内容给予反馈、提交任务。（）笔者在理解原始要求时，初步认为译者的报名和负责人的报名是同时进行的，不知是否理解有误）负责人的核心功能，并且也是对于整个平台而言比较重要的功能，就是对原始任务进行分割后重新形成“子任务”。笔者的初步构想是子任务继承于原始任务，并且其操作者为负责人指定的翻译者，通过这种方式，

使得子任务本身能够继承原始任务的必要成员信息，同时也能够派生新的成员（例如子任务的翻译内容、子任务的反馈信息等）负责人对于原始任务的操作权限次于发布者，对于子任务的操作权限次于翻译者，但负责人本身是原始任务和子任务之间的“媒介”。

任务翻译者对于任务的基本操作，主要是基于子任务而言。对于原始任务，笔者的设计是译者不可获取全文内容（但另一方面译者又可以通过译者互校获取其他译者的文本，所以感觉造成了一点矛盾），但可以对原始任务的基本信息进行查看。对于子任务的操作，包括：获取任务原文，翻译文本，暂存翻译内容，提交目前已经编辑的内容，查看负责人反馈，查看其他译者的任务内容，对其他译者的翻译任务给予反馈。译者的任务是整个平台需要实现的最基本功能，其中笔者欲实现多种语言的互相翻译，也是通过译者进行操作，另一方面，由于对于一个任务的操作通常是由多个译者共同进行，因此原始任务与子任务、原始任务与译者、子任务与译者，需要设计出比较好的对应关系以便于数据储存。

②.2 用户与用户的交互

用户与用户之间，除了普通用户应有的基本操作实现（例如查看用户信息，向用户发送消息，添加用户好友等），用户在同一个任务中的角色也影响了用户与用户的交互实现，并且在本平台中，用户与用户之间的交互，核心就在于同一个任务中不同角色、同种角色之间信息状态的交流。

②.2.1 用户与用户基本操作

笔者依据自己对于一个基本的多用户平台的基本功能，拟做出如下设计：用户查看其他用户基本信息、用户向用户发送消息、用户评价其他用户、用户添加用户为好友。

②.2.2 用户由于角色产生的权限操作

对于发布者，发布者具有查看报名负责人、报名译者的用户信息的权力，同时具有指定负责人的权力。

对于负责人，负责人同样能够查看报名用户信息，同时可以确定任务译者。

③平台功能综述

根据作业要求，总体而言，本平台需要实现以下基本功能：

- 用户可以通过唯一标识（Identity）注册并获取唯一账号
- 登陆后的用户可以发布任务、报名任务，但根据所申请的职位不同而存在不同的门槛要求
- 对于用户发布的任务，用户可以通过编辑编辑简介、设定任务报酬、设定任务周期等进行任务的完善
- 发布者可以从报名负责人的用户人选中确定合适的并且也是唯一的负责人负责任务后期的进行
- 负责人需要在原始任务的基础上确定译者报名的截止时间
- 并从报名译者的用户中选择合适的译者（可以有多个）参与任务翻译
- 确定译者后负责人需要对任务进行分割，并制定译者完成翻译任务
- 译者需要在任务周期时间内提交任务
- 负责人应针对译者提交的任务进行审核并提出反馈

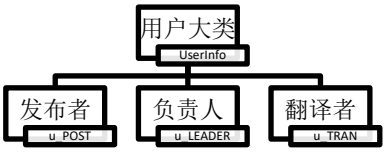
- 译者需要查看负责人的反馈并依据反馈做出修改
- 负责人整合译者的任务，最终提交完整任务
- 发布者检查任务后支付任务报酬
- 负责人与译者在发布者支付报酬后领取酬金。

## 1.1 总体功能描述

这部分需要将平台的功能按类别描述，可以采用逐条罗列或者表格的方式来阐述。

以下，笔者首先将主要的功能分为三个大类：用户大类，任务大类，及辅助功能大类（辅助功能即，与前述二者彼此关联，但是其功能是独立存在的）。

### 1.1.1 用户大类




| 用户个人功能 |
| --- |
| ● **用户注册**：通过该功能函数，用户可以以身份证号、手机号码或自定义昵称作为用户在平台中的唯一标识进行注册。注册函数需要实现对于身份证号、手机号码的检查，要存在获取当前平台所有用户的唯一标识的接口，并据此判断用户的注册名是否合法，若不合法，界面需要给出一定的提示。注册成功后，平台总用户需要增加该用户。同时系统自动为用户匹配一个唯一的账号。<br>● **用户完善个人信息**：通过该功能函数，用户可以完善自己的登录密码、个性头像、性别、生日、电子邮箱、以及对用户初始积分具有关键作用的语言资质描述。对于语言资质，存在平台自动识别关键词给予积分以及系统平台工作人员审核语言资质并给予积分两种方式，这一部分将作为平台管理功能进一步阐述。<br>● **用户登录**：用户注册成功并凭借系统匹配的账号以及自己初始完善个人信息时的用户密码能够登陆进入个人信息界面。<br>● **用户查看个人信息**：用户可以在个人信息页面直接查看自己的性别、生日、积分、余额等基本信息。<br>● **用户编辑个人信息**：在用户编辑个人信息页面，用户除了可以对基本信息进行修改，同时具有“转账”的功能。<br>● **用户充值**：用户可以通过输入所需金额对自己的账户余额进行充值。<br>● **用户更新个人数据**：由于存在数据储存，故用户应当在个人数据发生修改时相应对于更新队列进行补充。<br>● **用户发送消息**：用户可以编辑并向已经存在的用户发送消息。 |

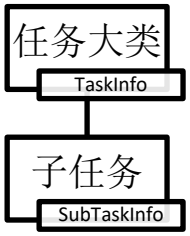
- **用户好友互动：**用户可以申请成为另一用户的好友，同样，接收到好友申请的用户可以拒绝好友申请，也可以同意成为好友。

| 用户与任务 | |
|---|---|
| - **用户查看任务：**（这里只认为是用户作为无关者对用户信息的查看）用户查看任务时，应该能够获得任务当前状态、任务简介、任务周期、任务酬金等基本信息，用户对于任务的基本操作是固定的，但受到任务状态和用户角色的限制，故查看函数需要提供足够的对外接口以便用户进一步对任务进行操作时具备足够的信息进行判断和限制。<br>- **用户发布任务：**所有用户都可以发布任务，但发布任务是需要对任务进行信息完善，包括上传待翻译书籍内容、指定翻译目标语言、指定翻译周期、确定任务酬金、确定负责人报名截止时间、编写任务简介。用户的发布任务函数需要覆盖到比较关键的错误并通过图形界面给予提醒。<br>- **用户报名负责人：**报名负责人的用户需要满足一定的积分条件，故对于积分不足的用户，当其选择报名时需要有一定的消息提醒。同时，由于笔者所设计的平台中，负责人与译者是可以在同一时间段报名的，故函数需要增加对于已经报名同一任务负责人、同一任务译者的相关提醒。即一名用户只能在同一个任务中担任一个角色。用户报名成功后，系统需要对发布者通过消息进行提醒。<br>- **用户报名译者：**用户报名译者不具有太多的限制，但若用户已经参与任务（如作为发布者、负责人、译者或已经报名的译者），则需要给予相应的提醒。用户报名成功后，系统需要对发布者通过消息进行提醒。若任务负责人已经确定，则系统也需要对任务负责人通过消息进行提醒。<br>- **用户个人推荐任务：**关于个性化推荐，笔者的设计思路是针对用户已参与任务的语言作为判断依据，并以此形成属于用户的兴趣任务队列。 | |
| 发布者 | - **发布者查看负责人报名信息：**作为发布者，当用户报名任务负责人成功后，系统会通过消息提醒任务发布者，发布者可以查看报名人的个人信息，包括语言资质证明、用户积分、用户已经参与负责的任务总数、用户已经参与翻译的任务总数等等，同时笔者增加了用户互相评论的功能，故负责人也可以查看其他用户对该用户的评价。 |
| | - **发布者确认负责人：**发布者确定负责人后，任务需要及时更新，包括任务状态更改为正在招募译者。同时，确认的负责人会收到招募消息。在笔者的设计中，允许发布者在截止时间之前确认负责人。 |
| | - **发布者查看任务：**发布者对于任务的查看，主要是在负责人提交任务以及任务完成后。负责人提交任务后，发布者可以对任务进行审核，进一步确定是否完成任务并给予酬金。（但理论上，同时也是笔者的设计，作为负责人有且只有一次任务提交机会）任务结束后，发布者则可以查看翻译后的任务全本。 |
| | - **发布者结束任务：**发布者可以直接点选结束当前任务，此时无论任务状态如何，将直接变更为结束状态，同时系统也会默认负责人、任务译者已经完成相应任务，从而实现负责人和译者的积分积累。（函数需要对是否存在负责人、译者进行判断） |
| | - **发布者支付酬金：**在笔者的设计中，发布者在发布任务伊始即已经支付任务酬金，系统为发布者暂存酬金，当发布者确认支付酬金 |

| | |
|---|---|
| | 时，系统将其按照比例转发给负责人与译者。另外，当发布者确认支付酬金时，任务状态都将直接变成结束状态。 |
| | ● |
| 负责人 | ● **负责人确认译者报名截止时间：**当发布者确认任务负责人后，负责人将会受到相应的消息提醒，同时负责人需要确定任务译者报名截止时间，若负责人不进行确定则无法进行后续操作。而若负责人未确定则系统默认为负责人报名截止时间。 |
| | ● **负责人查看译者报名信息：**作为负责人，当用户报名任务译者成功后，系统会通过消息提醒任务负责人，负责人可以查看报名人的个人信息，包括语言资质证明、用户积分、用户已经参与负责的任务总数、用户已经参与翻译的任务总数等等，同时笔者增加了用户互相评论的功能，故负责人也可以查看其他用户对该用户的评价。 |
| | ● **负责人确认译者：**负责人确定译者后，任务需要及时更新，包括任务状态更改为正在翻译。同时，确认的译者会收到招募消息。在笔者的设计中，允许负责人在截止时间之前确认译者。 |
| | ● **负责人分割任务：**在笔者的设计中，负责人对于译者并没有人数限制，当负责人确定分割任务时，此时的译者人数即是最终确定的译者人数。当译者人数为零食也会有相应的提醒。负责人在进行任务分割时，若某一个译者的任务为空，界面也应该出现提醒。 |
| | ● **负责人检查子任务：**负责人可以随时查看译者翻译的进行情况。需要将译者译文与原文进行对比。 |
| | ● **负责人给予译者反馈：**负责人针对译者提交的任务，可以提供反馈使译者进行修改。也就是说，译者所提交的任务内容，必须经过负责人审核、修改、整合后，才可能作为最终提交给发布者的任务。 |
| | ● **负责人暂存任务：**负责人可以暂存任务，可以对暂存任务和当前任务进行对比。 |
| | ● **负责人提交任务：**负责人确认提供任务，笔者的设计负责人仅有一次提交机会，发布者会进行检查并确定是否结束任务、支付酬金。 |
| | ● **负责人获得酬金：**负责人领取酬金时，需要判断当前任务发布者是否已经确认支付，确认支付后负责人可以直接领取相应比例的酬金。 |
| | ● |
| | ● |
| 译者 | ● **译者查看原始任务文本：**译者可以查看负责人为自己指定的子任务的文本内容。 |
| | ● **译者编辑任务：**译者可以根据文本内容和目标语言对文本进行翻译。 |
| | ● **译者暂存任务：**译者可以对已经编辑的内容暂存而不提交，并存在一定的接口查看上一次保存的内容。 |
| | ● **译者提交任务：**与负责人不同，译者具有多次提交任务的权限，但是译者需要根据每一次负责人给予的反馈对提交的文本进行修改后再次提交。 |
| | ● **译者查看反馈：**译者可以查看负责人以及其他译者针对自己提交的任务所给予的反馈。 |

| | |
|---|---|
| | ● **译者查看其他译者的任务并互相校对：**在笔者的设计中，译者互校的实现是通过译者的互相反馈实现的，译者不可以直接对其他译者的编辑内容进行修改，但可以通过反馈给予一定指点意见。同时，译者无法查看负责人以及其他译者给予对方译者的反馈内容。 |
| | ● **译者领取酬金：**负责人领取酬金时，需要判断当前任务发布者是否已经确认支付，确认支付后负责人可以直接领取相应比例的酬金。 |
| | ● |
| | ● |

### 1.1.2 任务大类




| 任务功能 |
|---|
| ● **获取任务状态：**由于在平台运行过程中，用户的操作与任务的状态是息息相关的，故需要一个借口获取任务状态。<br>● **获取任务支付状态：**负责人、译者是否能够领取酬金，取决于发布者是否确认支付，而这直接决定了任务的支付状态，故可以通过获取任务支付状态来做判断。<br>● **任务定位（获取当前任务）：**在平台运行过程中，随着用户操作的改变，用户当前处理的任务随时可能发生改变，所以需要一个定位功能实时获取用户当前处理的任务。<br>● **任务更新：**任务更新包括在种种操作后任务状态的更新，以及其内部成员，例如报名截止时间，例如译者名单等等的改变。<br>● **对任务书籍的处理：**在笔者的设计中，书籍作为一类，与其他成员组合而成为任务。用户进行任务发布时，需要具有一定的接口进行编辑。 |
| 子任务功能补充 |
| ● **子任务状态更新：**（多态）子任务同样需要更新，此时父类的更新函数作为虚函数存在。<br>● **获取子任务反馈：**任务反馈是针对子任务而言的，故这一部分需要另外储存，同时也只有作为子任务的指针类型才具有修改和访问权限。 |

### 1.1.3 辅助功能大类

#### 1.1.3.1 数据库

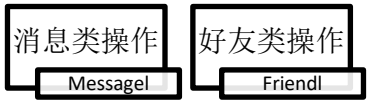

| 数据库存储取出数据<br>数据库存在分别能将数据存入 Mysql，以及存入内存的函数 |
| --- |
| ● **用户表**：存储用户基本信息，不包括用户信息中的数组型信息。具有唯一主键（useridentity）。<br>● **任务表**：存储全部任务信息，同样不包括数组型信息如译者组信息。具有唯一主键（taskname）。<br>● **消息表**：存储用户与用户之间、用户与系统之间的消息。具有唯一主键（sender+sendtime+receiver 的文本组合构成一个新的数据）。<br>● **任务与译者的中间关联表**：通过关联表实现任务与用户（译者）的二表查询。<br>● **任务与负责人报名人的中间关联表**：通过关联表实现任务与用户（负责人报名人）的二表查询。<br>● **任务与译者报名人的中间关联表**：通过关联表实现任务与用户（译者报名人）的二表查询。<br>● **子任务表**：存储全部子任务信息，具有唯一主键（subtaskname）。<br>● **子任务反馈表**：存储全部反馈信息，具有唯一主键（count）。<br>● **用户评价表**：存储全部用户互相评价的信息，具有唯一主键（count）。<br>● **用户朋友的中间关联表**：存储互相为朋友的用户 identity 信息，通过关联表实现用户与用户之间的二表查询。<br>● **用户添加朋友的中间关联表**：存储已经发送好友申请但并未成为好友的用户对 identity。通过关联表实现用户与用户之间的二表查询。 |

1.1.3.2 平台


| 系统平台类（登录模式下的操作） |
| --- |
| ● **平台登录**：笔者并未涉及平台的注册方法，以数据库中所包含的信息作为唯一标识进行登录。<br>● **平台查看用户信息**：平台可以查看所有用户的基本信息，尤其是语言资质证明。<br>● **平台给予用户积分**：平台可以根据用户撰写的语言资质证明给予一定积分。 |
| 系统平台类（自动模式下的操作） |
| ● **全局数组**：包括用户数组、任务数组、子任务数组、消息数组等，系统在用户进行相关的操作时需要进行更新，且在程序运行初始，数据库将储存的信息放入系统全局变量之中（内存），退出程序时，系统通过变化数组更新数据库的数据。 |

- **系统自动识别用户语言资质证明并给予积分：**在笔者的设计中，系统可以通过检索关键词对用户进行一定的积分评价，但此种方式相对来说对于用户的实际水平并没有完全考虑周全。
- **系统为用户匹配账号：**用户注册时，系统会为用户匹配唯一的账号，在笔者的设计中，匹配的账号就是用户在用户数组中的序列号。
- **系统消息提醒：**系统会对用户的新消息、新的好友申请进行提醒。用户所参与的任务状态发生改变时，以及用户成功参与某任务时，系统都会像用户发送相应的提示消息。

1.1.3.3. 消息与交友



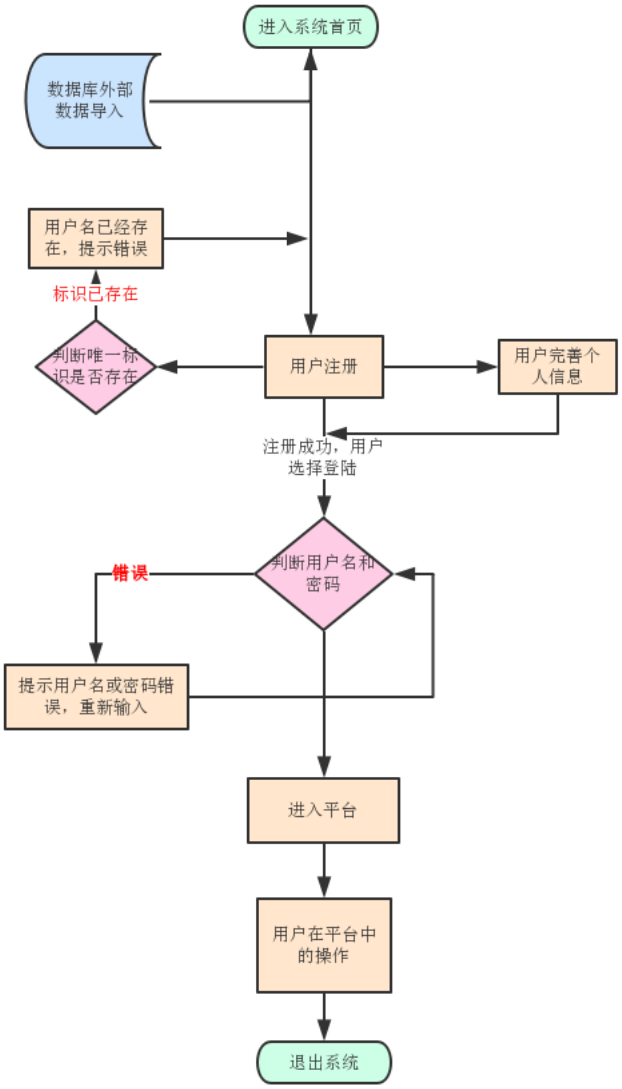

| 消息类 |
| --- |
| ● **消息构造：**一条消息应该包括消息内容、消息发送者、消息接收者、消息发送时间、消息是否已经阅读等成员，这些都可以通过构造消息类实现<br>● **用户发送消息：**用户可以编辑消息发送给指定用户。<br>● **系统消息提示：**用户所参与的任务状态发生改变时，以及用户成功参与某任务时，系统都会像用户发送相应的提示消息。 |
| 好友类 |
| ● **用户发送好友申请：**用户可以向指定用户发送好友申请。<br>● **用户添加好友：**接收到好友申请的用户可以选择添加对方为好友，互相为好友的用户可以相互查看彼此信息。<br>● **用户拒绝好友：**用户可以拒绝其他用户的好友申请。 |

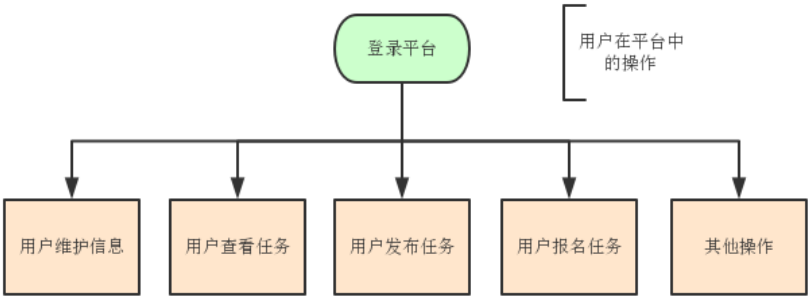
## 1.2 功能流程描述

结合上一节的总体功能和翻译任务的进行过程的不同状态描述对应的功能流程，可以用流程图配合文字来说明。

用户注册登录流程：

进入系统首页后，外部数据库数据导入。用户在拥有账号之前需通过身份证号、手机号或昵称进行注册，注册过程存在唯一性验证，若标识重复，用户需重新输入注册名。注册成功后，用户可以选择登陆系统，进入平台。用户在平台中的操作包括维护个人信息、查看任务、发布任务、报名任务以及其他一些基本操作。


进入系统首页
数据库外部数据导入
用户名已经存在，提示错误
标识已存在
判断唯一标识是否存在
用户注册
用户完善个人信息
注册成功，用户选择登陆
判断用户名和密码
错误
提示用户名或密码错误，重新输入
进入平台
用户在平台中的操作
退出系统

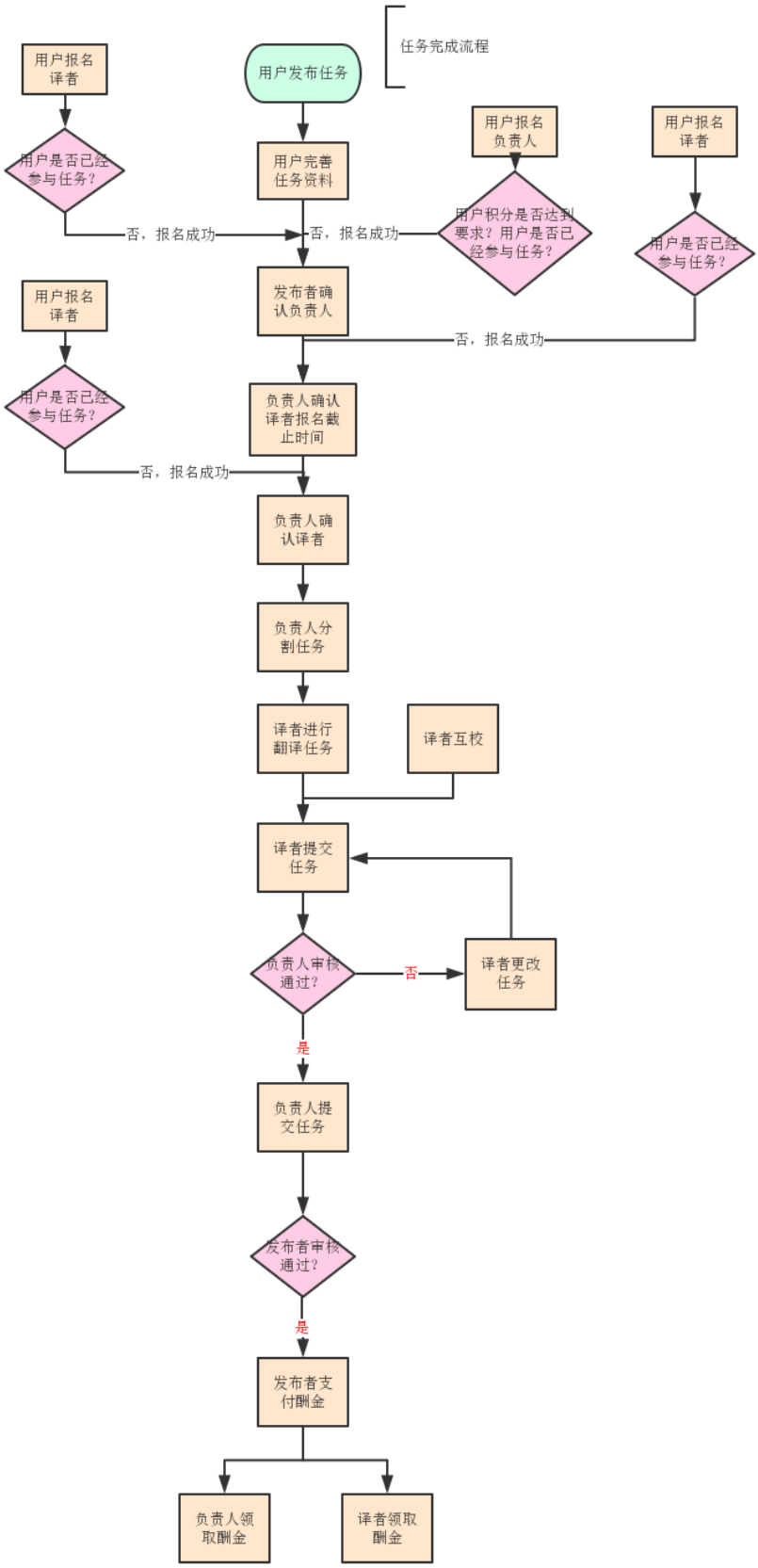



任务完成流程：在一个任务中，首先由发布者发布任务，发布者需要完成翻译任务文本的上传、目标语言的选择、翻译任务简介的填写、任务酬金的设定以及负责人报名截止时间的确定。任务发布后，感兴趣的用户可以报名成为负责人或者译者，但报名成为负责人需要一定的积分条件，同时报名的用户在本任务中并没有担任角色，也没有在此前报名。

发布者可以在截止日期后确认负责人，若希望提前确认，系统会进行一定的提醒。确认负责人后，负责人即确认译者报名截止时间，并在截止日期后确认译者，同样的，负责人也可以提前确认译者。

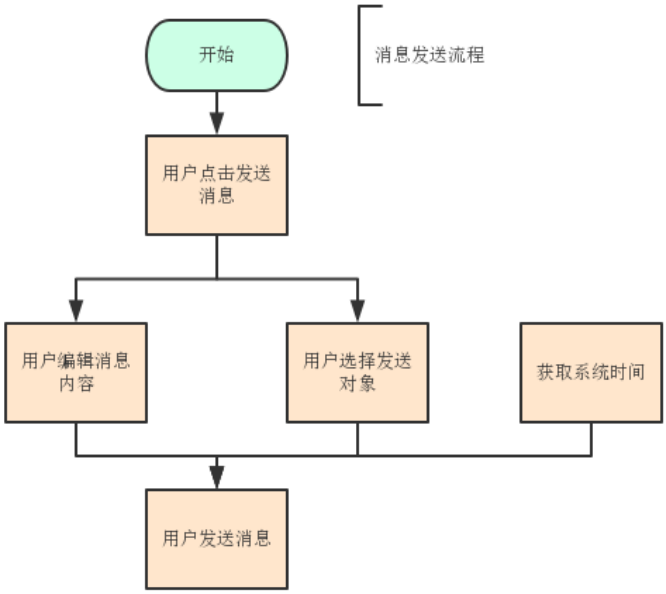
确认译者名单后，负责人开始进行任务分配，同时提醒译者在任务周期内完成任务。译者开始进行翻译任务，并且可以暂存、修改、提交任务，译者与译者之间可以相互评论进行翻译互校。译者提交任务后，负责人需要对译者提交的任务进行审核，并提出一定的反馈意见，译者收到反馈意见后，可以做出相应的修改，进行到负责人满意为止。

负责人确认译者翻译任务完成后，可以对各个译者的任务进行整合，并最终提交给任务发布者。任务发布者可以查看任务并选择结束任务、确认支付酬金。若发布者不确认支付，平台将代管发布者提交的任务酬金，而任务的负责人与译者也无法领取到酬金。


任务完成流程
用户报名译者
用户发布任务
用户报名负责人
用户报名译者
用户是否已经参与任务?
用户完善任务资料
用户积分是否达到要求?用户是否已经参与任务?
用户是否已经参与任务?
否，报名成功
否，报名成功
用户报名译者
发布者确认负责人
否，报名成功
用户是否已经参与任务?
负责人确认译者报名截止时间
否，报名成功
负责人确认译者
负责人分割任务
译者进行翻译任务
译者互校
译者提交任务
负责人审核通过?
否
译者更改任务
是
负责人提交任务
发布者审核通过?
是
发布者支付酬金
负责人领取酬金
译者领取酬金

消息发送流程：

用户点击发送消息后，用户需要制定消息发送对象、编辑消息内容，同时系统会自动获取当前的系统时间，信息汇总构造一条待发送的消息。



# 2 平台结构设计

这是进行复杂软件开发的第二步，即概要设计。此部分需要说清楚整个软件系统包含哪些模块（或功能部件），模块之间的关系和是怎样的；包含哪些主要的类，类之间的关系是怎样的（可以用 UML 类图或对象图表达）。
（此部分的子标题和结构自行拟定。）

## 2.1 类的设计

在笔者的设计中，程序主要包含以下类：

- UserInfo 类，派生 u_POST，u_LEADER，u_TRAN
- TaskInfo 类，派生 SubTaskInfo
- Server 类
- Message 类
- userComment 类
- Comment 类
- Book 类

主要分为四大模块：用户模块，任务模块，系统管理模块，数据存储模块。另外也包括一些小的模块：消息模块，反馈模块，用户评论模块。
用户模块包含用户对个人信息的维护操作，以及用户操作任务模块的接口，用户模块主要针对的是平台上的普通用户。
任务模块主要包含任务的构造和基本组成，所有对于任务的操作都直接体现在本模块中。任务模块同时包含子任务模块，指的是负责人为翻译者分配的任务，子任务模块则包含了用户（负责人与译者）对于子任务的操作。
系统管理模块包含管理员模块和系统自动管理模块。管理员模块中，可以通过登录平台管理员对用户数据进行更。在自动管理模式下，系统会在启动时从外部（数据库）获取所需要的数据并进行存储，在程序运行过程中，数据一经更改，系统会对相应更新（主要则是通过用户、任务的更新函数实现），同时系统也存在与数据库的接口将数据重新存储。
数据存储模块，主要指的是数据库与程序的数据交换。笔者将数据库的功能封装在一个头文件中，并通过相应的接口实现数据的存入取出。
小的模块则包含于或者是辅助于上述模块的一些功能。

### 2.1.1 用户模块

用户模块包含 UserInfo 类，TaskInfo 类，SubTaskInfo 类，Message 类，userComment 类。
UserInfo 类基本包含了用户模块的主要功能，例如用户登录、用户对于个人信息的维护、用户发布、报名任务等等。其派生的 u_POST，u_LEADER，u_TRAN 类，则是由于用户的角色区分而分别具有相应功能。
TaskInfo 类在用户模块中主要是作为任务数据的存储。用户对于任务的操作直接体现在该类中，用户与任务之间存在一种多对多的对应关系。派生类 SubTaskInfo 则存储的是子任务的相关数据。
Message 类，主要实现用户模块对于消息的操作。包括发送消息、查看消息等。

userComment 类主要存储用户互相评论的功能。

2.1.2 任务模块

任务模块包含 TaskInfo 类，SubTaskInfo 类，UserInfo 类，Comment 类，Book 类。其中 Book 类、Comment 类作为组件包含于任务模块之中。Book 主要存储的是任务书籍信息（或原始任务信息），Comment 类主要存贮的是子任务负责人与译者关于任务的交互信息，也即负责人和其他译者对于某译者的反馈信息。TaskInfo 类包含了任务模块所需要的全部数据成员与接口，包括任务更新、任务定位、任务构造，以及获取任务数据的相应接口。类似的 SubTaskInfo 作为其派生类，除了子任务所需要的数据外，其功能与父类基本相似。

UserInfo 类，的作用则是任务模块与用户模块进行标志识别，进而交互。

2.1.3 系统管理模块

系统管理模块中，系统管理员模块包含 Server 类，具有查看用户信息、修改用户积分的接口。

系统自动管理模块，则通过各类中的特性函数，在平台中直接实现，例如任务状态更新时自动对用户发送消息提醒，任务编辑出错时对用户进行错误提醒，等等。

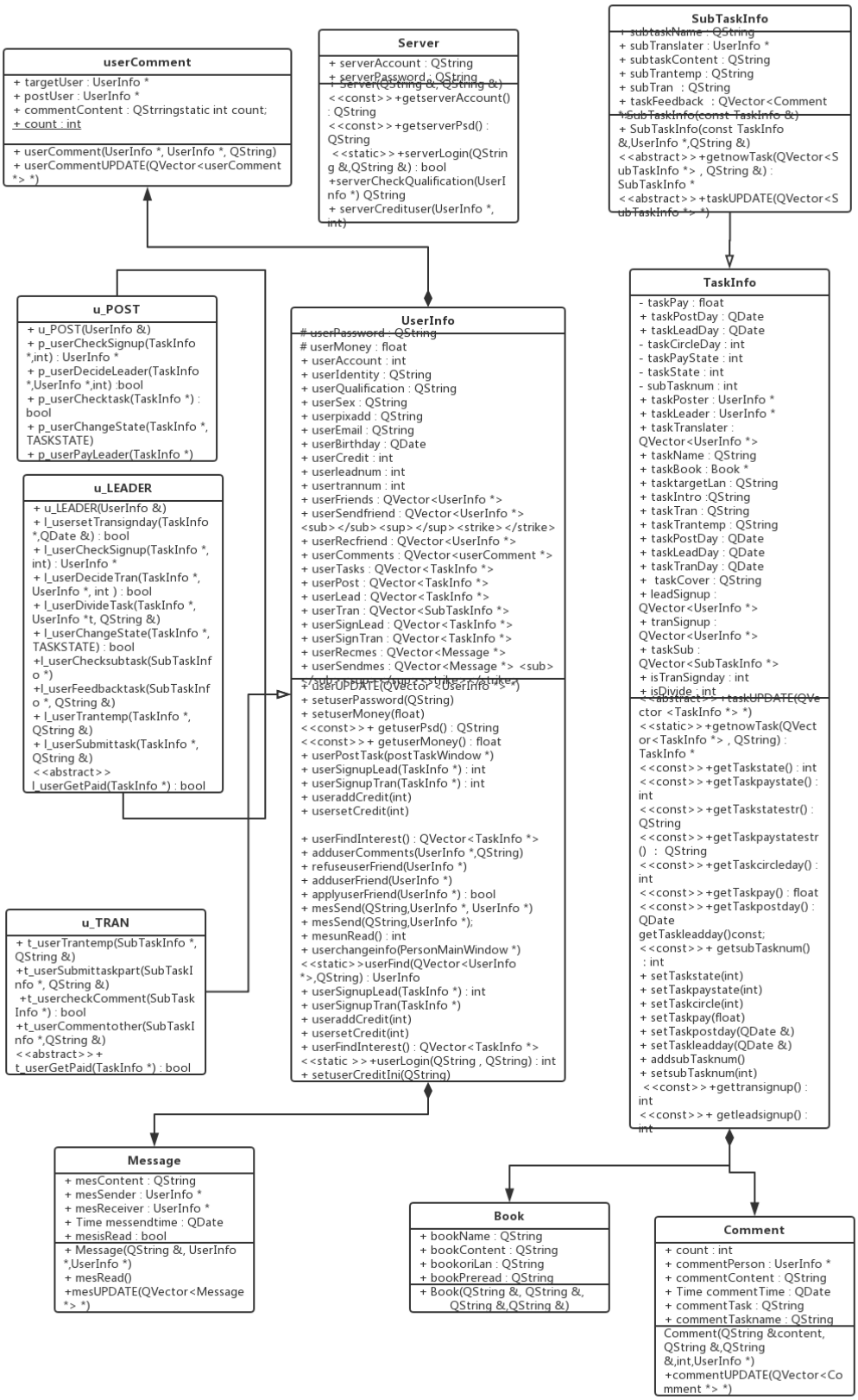
2.1.4 数据存储模块

在本次程序设计中，笔者是通过数据库实现数据离线存储。有关数据库的接口设计，即，数据的存入取出，将会通过数据存储模块实现。这一部分不以类的形式存在，而是以一个独立的头文件存在。

## 2.2 类的关系（UML 关系图）

在本次程序设计中主要包含 UserInfo 类，u_POST 类，u_LEADER 类，u_TRAN 类，TaskInfo 类，SubTaskInfo 类，Book 类，Comment 类，userComment 类，Server 类，Message 类。

其主要包含数据成员、函数成员以及类与类之间的关系如下：


userComment
+ targetUser : UserInfo *
+ postUser : UserInfo *
+ commentContent : QStrringstatic int count;
+ count : int
+ userComment(UserInfo *, UserInfo *, QString)
+ userCommentUPDATE(QVector<userComment *> *)
Server
+ serverAccount : QString
+ serverPassword : QString
+ Server(QString &, QString &)
<<const>>+getserverAccount() : QString
<<const>>+getserverPsd() : QString
<<static>>+serverLogin(QString &,QString &) : bool
+serverCheckQualification(UserInfo *) QString
+ serverCredituser(UserInfo *, int)
SubTaskInfo
+ subtaskName : QString
+ subTranslater : UserInfo *
+ subtaskContent : QString
+ subTrantemp : QString
+ subTran : QString
+ taskFeedback : QVector<Comment *>SubTaskInfo(const TaskInfo &)
+ SubTaskInfo(const TaskInfo &,UserInfo *,QString &)
<<abstract>>+getnowTask(QVector<SubTaskInfo *> , QString &) : SubTaskInfo *
<<abstract>>+taskUPDATE(QVector<SubTaskInfo *> *)
u_POST
+ u_POST(UserInfo &)
+ p_userCheckSignup(TaskInfo *,int) : UserInfo *
+ p_userDecideLeader(TaskInfo *,UserInfo *,int) :bool
+ p_userChecktask(TaskInfo *) : bool
+ p_userChangeState(TaskInfo *, TASKSTATE)
+ p_userPayLeader(TaskInfo *)
UserInfo
# userPassword : QString
# userMoney : float
+ userAccount : int
+ userIdentity : QString
+ userQualification : QString
+ userSex : QString
+ userpixadd : QString
+ userEmail : QString
+ userBirthday : QDate
+ userCredit : int
+ userleadnum : int
+ usertrannum : int
+ userFriends : QVector<UserInfo *>
+ userSendfriend : QVector<UserInfo *>
<sub></sub><sup></sup><strike></strike>
+ userRecfriend : QVector<UserInfo *>
+ userComments : QVector<userComment *>
+ userTasks : QVector<TaskInfo *>
+ userPost : QVector<TaskInfo *>
+ userLead : QVector<TaskInfo *>
+ userTran : QVector<SubTaskInfo *>
+ userSignLead : QVector<TaskInfo *>
+ userSignTran : QVector<TaskInfo *>
+ userRecmes : QVector<Message *>
+ userSendmes : QVector<Message *> <sub></sub><sup></sup><strike></strike>
+ userUPDATE(QVector <UserInfo *> *)
+ setuserPassword(QString)
+ setuserMoney(float)
<<const>>+ getuserPsd() : QString
<<const>>+ getuserMoney() : float
+ userPostTask(postTaskWindow *)
+ userSignupLead(TaskInfo *) : int
+ userSignupTran(TaskInfo *) : int
+ useraddCredit(int)
+ usersetCredit(int)
+ userFindInterest() : QVector<TaskInfo *>
+ adduserComments(UserInfo *,QString)
+ refuseuserFriend(UserInfo *)
+ adduserFriend(UserInfo *)
+ applyuserFriend(UserInfo *) : bool
+ mesSend(QString,UserInfo *, UserInfo *)
+ mesSend(QString,UserInfo *);
+ mesunRead() : int
+ userchangeinfo(PersonMainWindow *)
<<static>>userFind(QVector<UserInfo *>,QString) : UserInfo
+ userSignupLead(TaskInfo *) : int
+ userSignupTran(TaskInfo *)
+ useraddCredit(int)
+ usersetCredit(int)
+ userFindInterest() : QVector<TaskInfo *>
<<static >>+userLogin(QString , QString) : int
+ setuserCreditIni(QString)
TaskInfo
- taskPay : float
+ taskPostDay : QDate
+ taskLeadDay : QDate
- taskCircleDay : int
- taskPayState : int
- taskState : int
- subTasknum : int
+ taskPoster : UserInfo *
+ taskLeader : UserInfo *
+ taskTranslater : QVector<UserInfo *>
+ taskName : QString
+ taskBook : Book *
+ tasktargetLan : QString
+ taskIntro :QString
+ taskTran : QString
+ taskTrantemp : QString
+ taskPostDay : QDate
+ taskLeadDay : QDate
+ taskTranDay : QDate
+ taskCover : QString
+ leadSignup : QVector<UserInfo *>
+ tranSignup : QVector<UserInfo *>
+ taskSub : QVector<SubTaskInfo *>
+ isTranSignday : int
+ isDivide : int
<<abstract>>+taskUPDATE(QVector <TaskInfo *> *)
<<static>>+getnowTask(QVector<TaskInfo *> , QString) : TaskInfo *
<<const>>+getTaskstate() : int
<<const>>+getTaskpaystate() : int
<<const>>+getTaskstatestr() : QString
<<const>>+getTaskpaystatestr() : QString
<<const>>+getTaskcircleday() : int
<<const>>+getTaskpay() : float
<<const>>+getTaskpostday() : QDate
getTaskleadday()const;
<<const>>+ getsubTasknum() : int
+ setTaskstate(int)
+ setTaskpaystate(int)
+ setTaskcircle(int)
+ setTaskpay(float)
+ setTaskpostday(QDate &)
+ setTaskleadday(QDate &)
+ addsubTasknum()
+ setsubTasknum(int)
<<const>>+gettransignup() : int
<<const>>+ getleadsignup() : int
u_LEADER
+ u_LEADER(UserInfo &)
+ l_usersetTransignday(TaskInfo *,QDate &) : bool
+ l_userCheckSignup(TaskInfo *, int) : UserInfo *
+ l_userDecideTran(TaskInfo *, UserInfo *, int ) : bool
+ l_userDivideTask(TaskInfo *, UserInfo *t, QString &)
+ l_userChangeState(TaskInfo *, TASKSTATE) : bool
+l_userChecksubtask(SubTaskInfo *)
+l_userFeedbacktask(SubTaskInfo *, QString &)
+ l_userTrantemp(TaskInfo *, QString &)
+ l_userSubmittask(TaskInfo *, QString &)
<<abstract>>
l_userGetPaid(TaskInfo *) : bool
u_TRAN
+ t_userTrantemp(SubTaskInfo *, QString &)
+t_userSubmittaskpart(SubTaskInfo *, QString &)
+t_usercheckComment(SubTaskInfo *) : bool
+t_userCommentother(SubTaskInfo *,QString &)
<<abstract>>+
t_userGetPaid(TaskInfo *) : bool
Message
+ mesContent : QString
+ mesSender : UserInfo *
+ mesReceiver : UserInfo *
+ Time messendtime : QDate
+ mesisRead : bool
+ Message(QString &, UserInfo *,UserInfo *)
+ mesRead()
+mesUPDATE(QVector<Message *> *)
Book
+ bookName : QString
+ bookContent : QString
+ bookoriLan : QString
+ bookPreread : QString
+ Book(QString &, QString &, QString &,QString &)
Comment
+ count : int
+ commentPerson : UserInfo *
+ commentContent : QString
+ Time commentTime : QDate
+ commentTask : QString
+ commentTaskname : QString
Comment(QString &content, QString &,QString &,int,UserInfo *)
+commentUPDATE(QVector<Comment *> *)

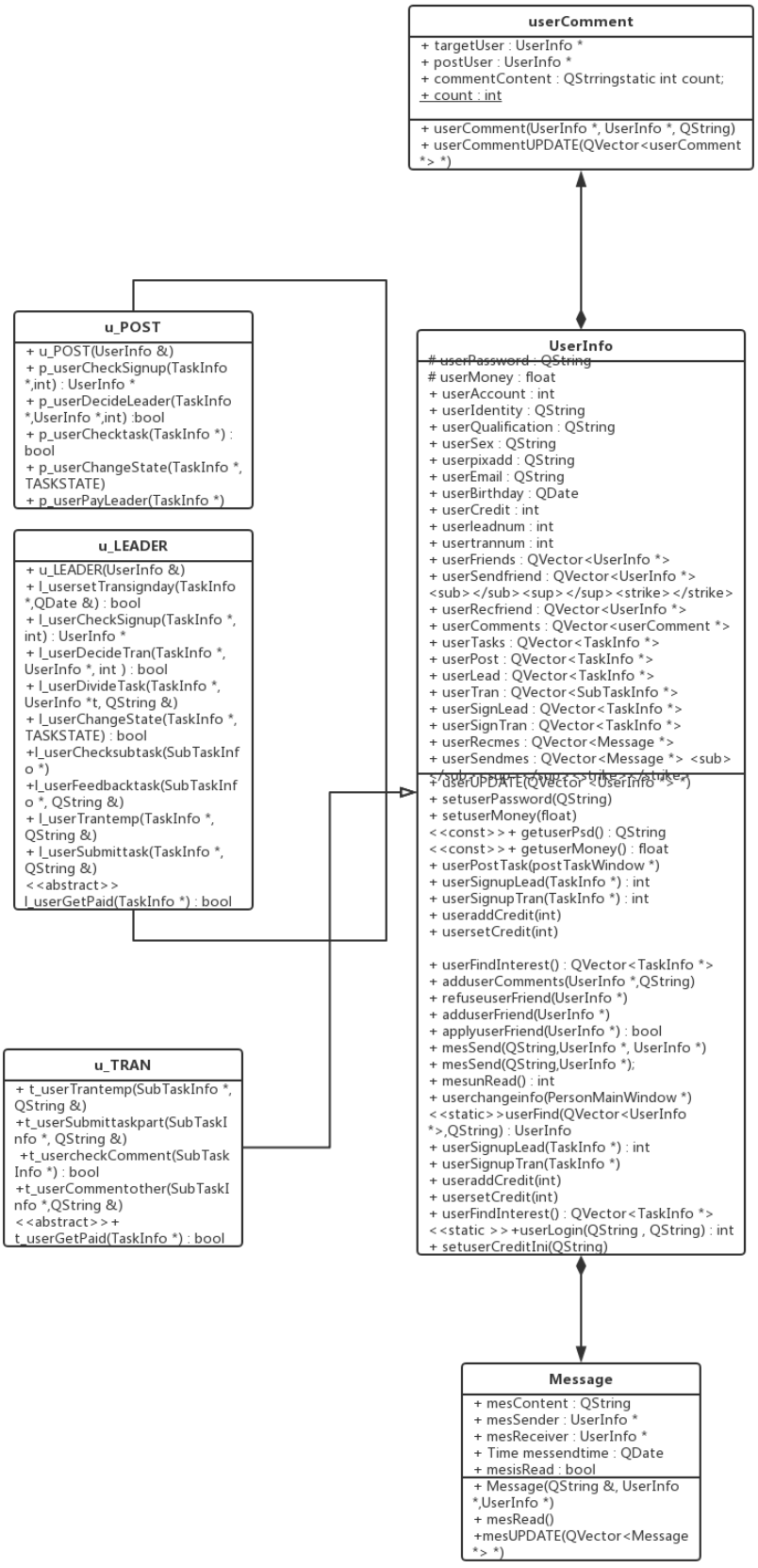
# 3 平台详细设计

这是进行复杂软件开发的第三步，即详细设计。此部分需要说清楚具体的模块是如何设计和实现的，类是如何具体实现的，类中的重点方法和算法是如何设计和实现的，数据库（或文件存储）的结构是如何设计的，界面是如何设计的，容错功能是如何设计的，以及各种设计思路等等，可以用图形或图表的方式加以说明。详细设计是编写代码前的最后一步设计工作，因而需要在需求分析和概要设计的基础上，说清楚所有需要在编码前明确的设计事项。

**注意：所有实现的加分功能需要单独详细说明。**

## 3.1 类结构设计

### 3.1.1 用户类（UserInfo）


userComment
+ targetUser : UserInfo *
+ postUser : UserInfo *
+ commentContent : QStrringstatic int count;
+ count : int
+ userComment(UserInfo *, UserInfo *, QString)
+ userCommentUPDATE(QVector<userComment *> *)
u_POST
+ u_POST(UserInfo &)
+ p_userCheckSignup(TaskInfo *,int) : UserInfo *
+ p_userDecideLeader(TaskInfo *,UserInfo *,int) :bool
+ p_userChecktask(TaskInfo *) : bool
+ p_userChangeState(TaskInfo *, TASKSTATE)
+ p_userPayLeader(TaskInfo *)
u_LEADER
+ u_LEADER(UserInfo &)
+ l_usersetTransignday(TaskInfo *,QDate &) : bool
+ l_userCheckSignup(TaskInfo *, int) : UserInfo *
+ l_userDecideTran(TaskInfo *, UserInfo *, int ) : bool
+ l_userDivideTask(TaskInfo *, UserInfo *t, QString &)
+ l_userChangeState(TaskInfo *, TASKSTATE) : bool
+l_userChecksubtask(SubTaskInfo *)
+l_userFeedbacktask(SubTaskInfo *, QString &)
+ l_userTrantemp(TaskInfo *, QString &)
+ l_userSubmittask(TaskInfo *, QString &)
<<abstract>>
l_userGetPaid(TaskInfo *) : bool
u_TRAN
+ t_userTrantemp(SubTaskInfo *, QString &)
+t_userSubmittaskpart(SubTaskInfo *, QString &)
+t_usercheckComment(SubTaskInfo *) : bool
+t_userCommentother(SubTaskInfo *,QString &)
<<abstract>>+
t_userGetPaid(TaskInfo *) : bool
UserInfo
# userPassword : QString
# userMoney : float
+ userAccount : int
+ userIdentity : QString
+ userQualification : QString
+ userSex : QString
+ userpixadd : QString
+ userEmail : QString
+ userBirthday : QDate
+ userCredit : int
+ userleadnum : int
+ usertrannum : int
+ userFriends : QVector<UserInfo *>
+ userSendfriend : QVector<UserInfo *>
<sub></sub><sup></sup><strike></strike>
+ userRecfriend : QVector<UserInfo *>
+ userComments : QVector<userComment *>
+ userTasks : QVector<TaskInfo *>
+ userPost : QVector<TaskInfo *>
+ userLead : QVector<TaskInfo *>
+ userTran : QVector<SubTaskInfo *>
+ userSignLead : QVector<TaskInfo *>
+ userSignTran : QVector<TaskInfo *>
+ userRecmes : QVector<Message *>
+ userSendmes : QVector<Message *> <sub></sub><sup></sup><strike></strike>
+ userUPDATE(QVector<UserInfo *> *)
+ setuserPassword(QString)
+ setuserMoney(float)
<<const>>+ getuserPsd() : QString
<<const>>+ getuserMoney() : float
+ userPostTask(postTaskWindow *)
+ userSignupLead(TaskInfo *) : int
+ userSignupTran(TaskInfo *) : int
+ useraddCredit(int)
+ usersetCredit(int)
+ userFindInterest() : QVector<TaskInfo *>
+ adduserComments(UserInfo *,QString)
+ refuseuserFriend(UserInfo *)
+ adduserFriend(UserInfo *)
+ applyuserFriend(UserInfo *) : bool
+ mesSend(QString,UserInfo *, UserInfo *)
+ mesSend(QString,UserInfo *);
+ mesunRead() : int
+ userchangeinfo(PersonMainWindow *)
<<static>>userFind(QVector<UserInfo *>,QString) : UserInfo
+ userSignupLead(TaskInfo *) : int
+ userSignupTran(TaskInfo *)
+ useraddCredit(int)
+ usersetCredit(int)
+ userFindInterest() : QVector<TaskInfo *>
<<static >>+userLogin(QString , QString) : int
+ setuserCreditIni(QString)
Message
+ mesContent : QString
+ mesSender : UserInfo *
+ mesReceiver : UserInfo *
+ Time messendtime : QDate
+ mesisRead : bool
+ Message(QString &, UserInfo *,UserInfo *)
+ mesRead()
+mesUPDATE(QVector<Message *> *)

### 3.1.1.1 属性

```
protected:
    QString userPassword;        //用户密码
    float userMoney;             //用户余额
public:
    int userAccount;             //用户账号
    QString userIdentity;        //用户昵称(uni)
    QString userQualification;   //用户语言资质
    QString userSex;             //用户性别
    QDate userBirthday;          //用户生日
    QString userEmail;           //用户邮箱
    QString userpixadd;          //用户头像路径
    int userCredit;              //用户积分
    int userleadnum;             //用户负责的任务总数
    int usertrannum;             //用户翻译的任务总数
    //**********待补充
    QVector<UserInfo *> userFriends;          //用户好友
    QVector<UserInfo *> userSendfriend;       //用户发出好友请求
    QVector<UserInfo *> userRecfriend;        //用户收到好友请求
    QVector<userComment *> userComments;      //用户评价标签
    QVector<Message *> userRecmes;            //用户收到的消息
    QVector<Message *> userSendmes;           //用户发送的消息
    QVector<TaskInfo *> userTasks;            //用户任务列表
    QVector<TaskInfo *> userPost;             //用户发布的任务
    QVector<TaskInfo *> userLead;             //用户负责的任务
    QVector<SubTaskInfo *> userTran;          //用户翻译的任务
    QVector<TaskInfo *> userSignLead;         //用户报名成为负责人的任务
    QVector<TaskInfo *> userSignTran;         //用户报名成为译者的任务
```

### 3.1.1.2 功能实现

- 用户注册时对于 userIdentity 的识别与判断：
  userIdentity 作为用户的唯一标识，如何进行 userIdentity 的唯一性检查是非常关键的，同时用户在注册时，由于需要选择手机号、身份证号作为注册标志，故需要做类型检查。这一部分的设计，笔者主要通过 if 条件判断以及 for 循环寻找是否存在相同 Identity 用户。用户注册时，若检查发现用户输入格式有误，系统将会报错；若检查发现用户标识已经存在，系统也会提出相应的提醒。
- 利用 userIdentity 的唯一性寻找定位用户
  在笔者的设计中，系统存在一个全局变量<UserInfo *>NOW_USER 作为当前操作用户的用户指针。在每一次用户登入系统时，一方面系统会利用 Identity 的唯一性寻找到用户地址并标记。另一方面，在程序运行过程中，当用户查看其他用户信息时，也需要利用 userIdentity 进行定位，故笔者设计了 <<static>>userFind(QVector <UserInfo *>, QString ) : UserInfo 这样一个方法，随时可以调用定位用户。
- 设置用户初始积分
  用户的初始积分有两种计算方法：一种是利用用户填写的 userQualification，系统调用 setuserCreditIni 的方法直接计算。笔者采用在 QString 数组中（也就是 userQualification）

中是否存在某些关键字（例如：tofel、ielts、英语、gre）等，根据不同关键词出现的频次进行累计打分。另一种方法是平台管理员检查用户填写的语言资质证明，并主观判断设定分数，这需要系统管理员登录平台进行操作，通过 Server 类实现。

- 用户信息更新

  由于在程序设计中，数据最终需要利用数据库进行储存，为了减轻数据库存储数据的负担，笔者设计了全局变量 QVector <UserInfo *>CHANGE_USER 用于属性发生改变的用户。对应的方法是静态成员函数 userUPDATE(QVector<UserInfo *> *)，调用此方法的对象会将自身的地址插入传入参数数组地址的指针列表之中。

- 用户信息对外接口

  在本次程序设计中，存在的比较大的不足之一是由于在早期设计是考虑不够完善，大量的属性作为类的共有成员暴露在外，无法实现很好的封装效果。其中，用户的余额 userMoney、密码 userPassword 作为保护属性，需要对应的接口取得，在程序设计中将 getuserMoney 与 getuserPsd 作为常成员函数。返回用户的余额与密码数值。

- 用户消息盒子操作

  在类的设计中，Message 类作为 UserInfo 类的组件，构成了用户的“消息盒子”。用户对于消息类有以下操作：阅读消息和发送消息（同样是 get 与 set）。首先，消息类包含 bool 类型属性 mesIsRead，标记用户是否阅读消息。用户登录时，系统会判断并标记用户消息盒子中的未读消息（也就是 mesIsRead == false），并通过消息弹窗提醒用户。

  用户阅读消息时，系统将会获取用户的消息内容、消息发送者、消息发送时间，并向用户呈现。

  用户任务状态发生改变时，消息盒子需要向用户发送提醒。在笔者的设计中，重载了 mesSend 函数，实现了多态。mesSend 方法具有两种不同的调用参数列表：mesSend(QString &content,UserInfo *receiver, UserInfo *sender), 以及 mesSend(QString &content,UserInfo *receiver)，均无返回值。具有第三个参数 sender 的方法表示用户向其他用户发送消息，不具有第三个参数的方法则表示系统发送给用户的消息。通过函数重载的方式，提高了方法的利用效率。而用户向其他用户发送消息，也是笔者所涉及的加分功能之一。

- 用户发布任务

  用户发布一个任务，需要完善任务的基本信息，在这之中，包括任务的原始文本内容（或者说待翻译的书籍），即 TaskInfo 类成员 Book，同时也要编辑任务的任务简介（taskIntro），任务目标语言（tasktargetLan），任务周期（taskCircleDay），任务酬金（taskPay），任务负责人报名截止时间（taskPostDay）等。

  首先，为了方便取用各类数据，笔者将用户发布任务的方法生命为发布任务窗口界面 PostTaskWindow 的友元函数，直接获取用户输入的数据信息。

  第二，用户正式发布任务之前，需要对用户的输入进行检查，例如：发布时间是否有误？发布人是否有足够的余额支付任务酬金？发布者是否已经上传任务简介？等等，在用户发布任务方法中都进行了相应的错误检查，保证了程序在一定范围内的容错性

- 用户报名负责人

  当用户报名负责人时，需检查用户积分是否达到负责人标准，用户是否已经成为任务的参与人（如用户是发布者），用户是否已经报名成为负责人或者译者。而对于不同的错误，系统的提醒应该是不一致的，所以在这里设计了一个返回值为 int 类型的方法：

  int userSignupLead(TaskInfo *task)

  同时设计了

  EnumSignLead{l_Success=0,l_LackCredit,l_AlreadySignup,l_Overdue,l_Poster,l_AlreadySignu

pTran};
作为结果返回值，以便系统进行判断并给出相应的提示。

- 用户报名译者
  当用户报名译者时，需检查是否已经成为任务的参与人（如用户是发布者、负责人），用户是否已经报名成为负责人或者译者。而对于不同的错误，系统的提醒应该是不一致的，所以在这里设计了一个返回值为 int 类型的方法：
  int userSignupTran(TaskInfo *task)
  并设计了

enumSignTran{t_Success,t_AlreadySignup,t_Overdue,t_Poster,t_Leader,t_AlreadySignupLeader};
作为结果返回值，以便系统进行判断并给出相应的提示。

### 3.1.1.3 派生类

UserInfo 作为父类，派生了三个子类：u_POST,u_LEADER,u_tran。
以下将从三个子类派生出的新方法出发简单谈一谈笔者的设计思想。

U_POST 类（发布者类）

- 发布者查看负责人报名信息
  发布者可以查看已经报名的负责人，查看负责人的基本信息，包括用户积分，用户曾经参与过的任务数量等等。
- 发布者确认负责人
  发布者严格上来说必须在截止日期之后才能确认负责人，但笔者为了方便调试，所以允许发布者在截止日期之前确认负责人，但是系统需要给出相应的提示。
- 发布者修改任务状态
  发布者具有直接将任务结束的权限。尽管在进行函数设计时，笔者进行了输入状态的分类，但理论上即使作为发布者，也仅仅具有发布任务时系统自动将任务状态修改为“正在招募负责人”，以及在负责人提交任务之后结束任务，改变任务状态为“任务结束”，其他的状态修改都应该由系统完成。
- 发布者查看任务
  在笔者的设计中，发布者的界面对于任务信息的读取是最多的，发布者可以查看负责人报名信息、译者报名信息、任务当前负责人、任务当前译者，其他的信息也包括任务名字、任务书籍、任务状态、任务酬金等等。此为任务的基本查看界面。
  当负责人提交任务后，负责人应该能够查看已经提交的任务，从而进行审核并决定是否结束任务，给予报酬。
  当发布者结束任务后，发布者能够直接查看到最终提交的译文以及原文内容。
- 发布者支付酬金
  发布者在发布任务时已经支付酬金，此时酬金由系统暂存。而发布者具有确认支付酬金的接口，一经确认，负责人与译者能够领取任务酬金。

U_LEADER 类（负责人类）

- 负责人确认译者报名截止日期
  负责人确认后，可以进入相应的任务窗口确认译者报名的截止时间。在笔者的设计中，译者是在整个任务进行过程中（且在发布者确认截止日期前）都能够报名参与任务的，在负责人确认截止日期之前，系统默认截止日期为发布者设定的负责人报名截止信息。
- 负责人查看译者报名信息
  负责人可以查看译者的报名信息，包括译者的积分、译者的评价、译者参与过的任务总数等等。
- 负责人确认译者
  负责人原则上应该是在译者报名截止日期之后才能确认译者名单。但笔者为了方便调试，允许负责人在截止日期之前确认译者，但是系统需要给出相应的提示信息。负责人确认译者并没有人数的限制（即，负责人可以在选择阶段再确定自己需要多少译者）。
- 负责人分割任务
  负责人在确认译者后，需要对原始任务进行分割（主要是文本内容的分割）。负责人的这一方法将任务分割成为任务的派生类 SubTaskInfo,其中既包含原始任务的数据，同时也具有新的属性（例如，子任务对应的译者，子任务对应的文本内容等等）。负责人在分割任务时，需要遵循有多少译者分割多少任务的原则。故一方面，在方法的是线上，笔者采用了动态数组的方式，利用 for 循环为译者分配任务。在界面的呈现上，笔者同样是利用动态数组 QLineEdit 动态呈现译者子任务，然后由负责人进行编辑。当负责人没有给某一个译者分配任务时，也就是子任务任务内容为空时，负责人会收到系统提醒。
- 负责人检察任务
  负责人在分配任务后，需要具有检查任务的接口。通过这一接口，负责人可以实时跟踪译者已经提交的译文内容，并作出反馈。
- 负责人给予译者反馈
  负责人在审核译者已经提交的译文时，可以对译文内容作出反馈。需要注意的是，译者的任务提交不具有次数显示，同时也随时可以进行更改。译者需要依据负责人的反馈进行修改。
- 负责人暂存任务
  负责人收到提交的任务后可以暂时保存任务。
- 负责人提交任务
  负责人具有提交最终任务的接口。在笔者的设计中，由于具有全局变量 NOW_TASK，故负责人可以通过将子任务中的译文内容进行整合（也即是作为负责人端的输入），然后改变 NOE_TASK 相应的（译文）属性。
- 负责人获得酬金
  在发布者确认支付酬金后，负责人可以领取酬金。但在发布者未确认之前，负责人是无法通过这一方法领取到酬金的，换句话说，本方法存在一个判断。因而笔者在设计任务类时，设计了一个用于表示任务是否支付的 taskPayState 属性，用于标记任务发布者是否已经确认支付。

U_TRAN 类（译者类）

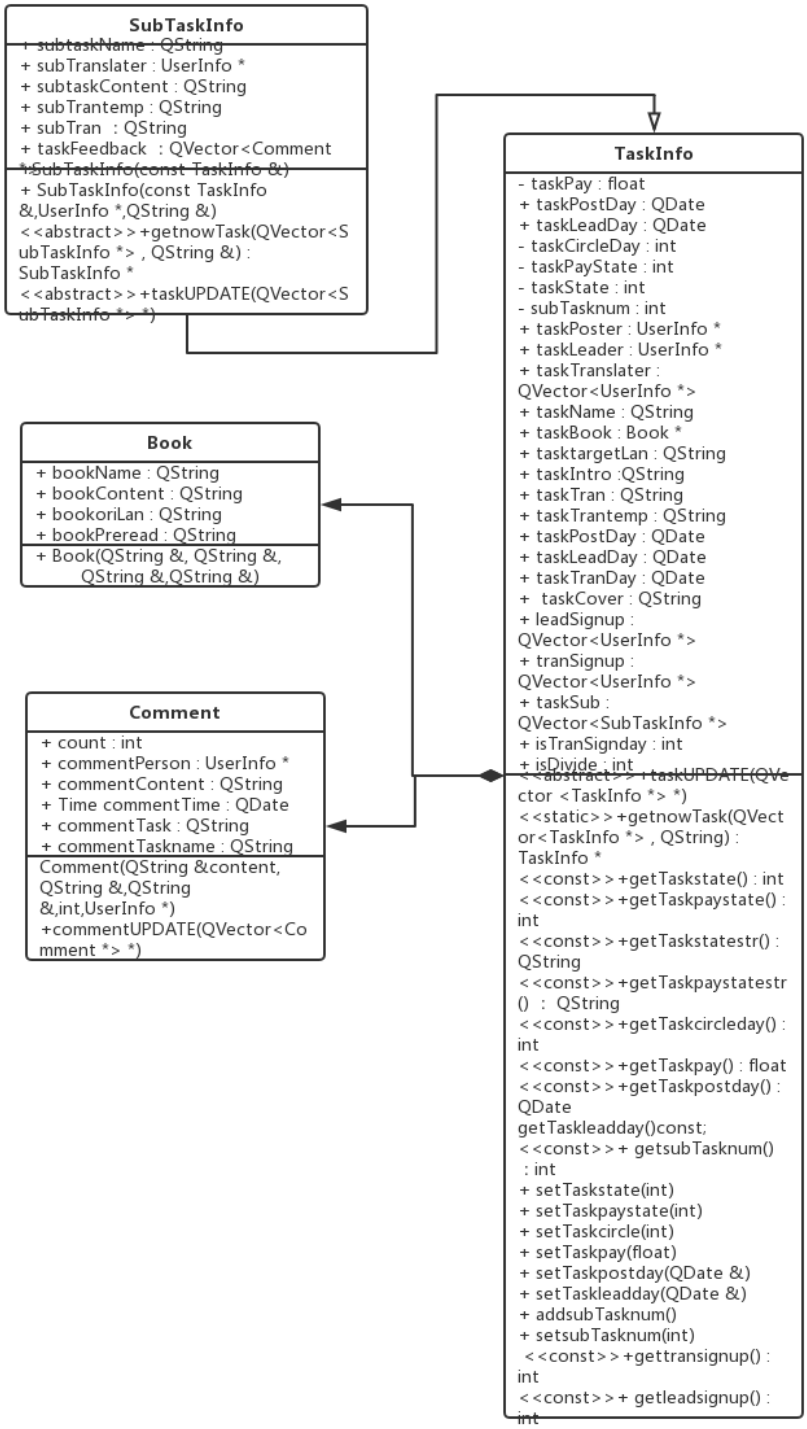
- 译者暂存任务
  译者能够暂时保存已经完成的译文而不提交。笔者在设计 SubTaskInfo 时，设计了一个 QString 类型的 SubTrantemp 的属性，用于存放目前译者保存的译文同时笔者也设计了查看上一次保存的按钮触发事件，译者可以通过这一方法获取上一次保存的译文信息。对于译者保存的译文信息，发布者、负责人是不具有查看权限的，因而这一方法也是

u_TRAN(译者类)所派生的方法。

- 译者提交任务
  译者具有提交任务的接口，与负责人的差异在于，译者具有多次提交的机会。提交后的译文会保存在 SubTaskInfo 的属性 SubTran 之中，而负责人也具有查看这一属性的方法。
- 译者查看反馈
  译者可以查看负责人对于译者已经提交的反馈的接口。笔者将反馈设计成为单独的类 Comment，其中包含计数属性 count，以及反馈提出者（负责人），反馈针对的子任务、反馈针对的译文，反馈的具体内容以及提出反馈的时间。对于反馈针对的译者，是能够查看到反馈的发出者和发出时间、发出内容，而同一任务下的其他译者则不具有查看负责人对于该译者的反馈信息。笔者通过动态数组存储反馈。
- 译者互校
  译者互校是笔者设计中的附加功能。其实现方法是译者之间可以相互查看彼此的译文的原文，并提出反馈意见。需要注意的是，其他译者并不具有修改译者译文的权限，仅能通过反馈的形式提出意见。同样的，译者的反馈也是通过动态数据存取，而以反馈提出者作为标记。
- 译者领取酬金
  当任务发布者确认支付酬金后，译者可以领取酬金。但在发布者未确认之前，译者同样是无法通过这一方法领取到酬金的，换句话说，本方法存在一个判断。因而笔者在设计任务类时，设计了一个用于表示任务是否支付的 taskPayState 属性，用于标记任务发布者是否已经确认支付。

### 3.1.2 任务类（TaskInfo）

#### 3.1.2.1 属性


SubTaskInfo
+ subtaskName : QString
+ subTranslater : UserInfo *
+ subtaskContent : QString
+ subTrantemp : QString
+ subTran : QString
+ taskFeedback : QVector<Comment *>
+SubTaskInfo(const TaskInfo &)
+ SubTaskInfo(const TaskInfo &,UserInfo *,QString &)
<<abstract>>+getnowTask(QVector<SubTaskInfo *> , QString &) : SubTaskInfo *
<<abstract>>+taskUPDATE(QVector<SubTaskInfo *> *)
TaskInfo
- taskPay : float
+ taskPostDay : QDate
+ taskLeadDay : QDate
- taskCircleDay : int
- taskPayState : int
- taskState : int
- subTasknum : int
+ taskPoster : UserInfo *
+ taskLeader : UserInfo *
+ taskTranslater : QVector<UserInfo *>
+ taskName : QString
+ taskBook : Book *
+ tasktargetLan : QString
+ taskIntro :QString
+ taskTran : QString
+ taskTrantemp : QString
+ taskPostDay : QDate
+ taskLeadDay : QDate
+ taskTranDay : QDate
+ taskCover : QString
+ leadSignup : QVector<UserInfo *>
+ tranSignup : QVector<UserInfo *>
+ taskSub : QVector<SubTaskInfo *>
+ isTranSignday : int
+ isDivide : int
<<abstract>>+taskUPDATE(QVector <TaskInfo *> *)
<<static>>+getnowTask(QVector<TaskInfo *> , QString) : TaskInfo *
<<const>>+getTaskstate() : int
<<const>>+getTaskpaystate() : int
<<const>>+getTaskstatestr() : QString
<<const>>+getTaskpaystatestr() : QString
<<const>>+getTaskcircleday() : int
<<const>>+getTaskpay() : float
<<const>>+getTaskpostday() : QDate
getTaskleadday()const;
<<const>>+ getsubTasknum() : int
+ setTaskstate(int)
+ setTaskpaystate(int)
+ setTaskcircle(int)
+ setTaskpay(float)
+ setTaskpostday(QDate &)
+ setTaskleadday(QDate &)
+ addsubTasknum()
+ setsubTasknum(int)
<<const>>+gettransignup() : int
<<const>>+ getleadsignup() : int
Book
+ bookName : QString
+ bookContent : QString
+ bookoriLan : QString
+ bookPreread : QString
+ Book(QString &, QString &, QString &,QString &)
Comment
+ count : int
+ commentPerson : UserInfo *
+ commentContent : QString
+ Time commentTime : QDate
+ commentTask : QString
+ commentTaskname : QString
Comment(QString &content, QString &,QString &,int,UserInfo *)
+commentUPDATE(QVector<Comment *> *)

### 3.1.2.2 功能实现

- 任务的唯一标识 taskName
  在笔者的设计中，每一个任务都存在唯一标识，即发布者发布的任务书籍名称 BookName 与发布者的表示 userIdentity 共同构成的字符数组。通过此唯一标识，用户可以查找任务、系统也能够实现任务定位。
- 任务的状态获取
  任务的状态属性，主要包括任务的进行状态，这也是作业要求中必有的，包括
  正在招募负责人（WaitingLeader），正在招募译者（WaitingTranslaters），正在进行翻译（Translating），任务结束（Finished）。随着用户操作的进行，任务的状态会发生相应改变。笔者通过 enum 创建了一个任务状态的别名。
  在笔者的设计中，首先状态改变是系统默认进行，即，例如当用户发布任务成功，系统便会调用 setTaskstate(int)函数，改变任务状态。
  另一方面，由于任务发布者具有结束任务的操作，所以对于任务发布者也提供了这样的接口，发布者可以直接修改任务状态，但一般情况下，只能结束任务。
  任务的其他状态属性还包括：
  任务译者报名截止时间确认状态。笔者设计了 enum TRANDDLSTATE{NOTSET, SET}进行标记，当负责人设定译者报名截止时间时，一方面系统会判断负责人是否已经设定，另一方面负责人设定成功后其状态也会发生改变。
  任务的任务分割状态，也就是负责人是否已经分割子任务。同样的，笔者设计了 enum DIVIVDESTATE{NOTDIVIDE, DIVIDE}进行标记。
  任务的支付状态。也就是任务发布者是否已经确认支付。笔者是利用 enum PAYSTATE{NOTPAY, PAY}进行标记。当发布者未确认支付时，其状态默认为 NOTPAY，此时负责人与任务译者不能够领取到任务酬金。当发布者确认支付，状态修改为 PAY，此时负责人和译者才能够领取到酬金之中属于自己的份额。
- 任务的状态获取
  任务的更新，与用户数组的更新类似。笔者采用了全局动态数组的模式存储当前系统的任务数据，当某任务信息或状态发生改变时，系统会通过调用 taskUPDATE 方法，自动更新 CHANGE_TASK 数组，实现变化的保存，进而更新数据库数据。
- 任务文本的设计
  在笔者的设计中，由于需要翻译的书籍本身，其实就是一个有独立属性与方法的模块，所以笔者单独设计了 Book 类，作为任务的原始数据。
  Book 包含以下属性：

QString bookName;表示任务待翻译书籍的名称，由发布者手动编辑。
QString bookContent;表示任务待翻译的内容，由发布者上传。
QString bookoriLan;表示任务待翻译文本的原始语言，由发布者手动编辑。目前平台具有英语、中文、法语三种语言选项。
QString bookPreread;表示书籍的预读内容，原本是希望作为译者报名试译的部分，但可惜未能实现。

- 任务反馈的实现
  任务反馈主要体现为子任务的反馈意见，在笔者的设计中，任务反馈包含负责人的反馈，也包含其他译者的反馈意见。需要注意的是，收到反馈的译者以及相应的负责人具有查看所有反馈的权力，而其他译者则仅仅具有查看自己发出的反馈的权限。因而在笔者的

设计中，对于译者与负责人的查看反馈构成了两个函数（此处并没有设计成多态，应该也是可以的）。对于译者会进行判断。

由于反馈本身同样自成模块，故笔者同样是单独设计了反馈类 Comment。

其包含的属性有：

int count;用于标记反馈的序列位置，不具有内容上的实际意义，仅仅作为数据库的自增约束键。

UserInfo *commentPerson;反馈的发出者，可能是任务发布人，也可能是任务其他译者。

QString commentContent;反馈的实际内容，即用户端的输入内容。

QDateTime commentTime;反馈的时间，通过系统自动获取系统时间。

QString commentTask;反馈的任务内容，即子任务的所提交的内容。

QString commentTaskname;反馈的子任务名称，也是子任务的唯一标识。

Comment 类内部实现的方法有：

Comment 的构造，需要输入的参数包括反馈的文本内容、反馈的子任务名称、反馈的子任务内容、以及根据当前系统全局变量反馈数组 QVector<Comment *> 的数组大小而自动增加的反馈序列号。

```
Comment(QString &content, QString &taskname, QString &taskcontent,int count,UserInfo *person);
```

同时还有 Comment 的自动更新方法，与用户全局数组、任务全局数组类似，通过 CHANGE_COMMENT 的动态数组存储新增加的以及发生动态变化的任务反馈。

```
void commentUPDATE(QVector<Comment *> *commentlist);
```

Comment 在用户模块中的实现方法：

```
void l_userFeedbacktask(SubTaskInfo *subtask, QString &feedback);
```

此为负责人给予反馈的方法，由负责人类对象 u_LEADER 调用，传入的参数即相应的子任务以及反馈内容。

```
bool t_usercheckComment(SubTaskInfo *task);
```

此为译者查看任务反馈的方法，由译者 u_TRAN 调用。可以查看译者所属子任务的反馈，包括反馈发出时间、反馈发起的对象、反馈的内容等。

```
void t_userCommentother(SubTaskInfo *task,QString &comment);
```

此为译者对其他译者的反馈方法，由译者 u_TRAN 调用。译者能够通过此方法实现译者互校，这一附加功能将于后文详细阐述。

●

### 3.1.2.3 派生类

TaskInfo 作为父类，派生了子类：SubTaskInfo。

相比父类，SubTaskInfo 具有如下新增加的属性：

QString subtaskName;此为子任务名称，同时也是子任务的唯一标识，构造方法为原始任务名称与翻译者标识的组合。

UserInfo *subTranslater;此为子任务所分配的译者用户指针。 此属性主要是用于数据存放时用户与任务数组的对应。

QString subtaskContent;此属性为子任务文本内容，即负责人为译者划分的翻译部分。
QString subTrantemp;此属性存放的是子任务译者暂存的译文内容，译者须有查看上一次暂存的接口进行访问。
QString subTran;此属性是译者提交的译文内容，译者和负责人均需要具有查看这一内容的接口。
QVector<Comment *> taskFeedback;这一部分存放的是子任务所收到的反馈信息，是笔者单独设计的 Comment 类指针动态数组。

派生的 SubTaskInfo，一方面是由负责人分割任务时产生的，另一方面，由译者用户和负责人用户使用相应接口进行操作。为了方便用户操作进行，笔者采用了子任务唯一标识 subtaskName 用于子任务的定位，并且设计了全局变量 NOW_SUB 与 CHANGE_SUB，ALL_SUBTASK，前者用于存放当前访问的子任务指针，后者分别用于存放数据变化的子任务和系统所有子任务。
另外，我们容易分析得到，译者对于任务的主要操作是基于子任务进行的，因而用户译者模块具有更多的和子任务交互的接口，而负责人也需要具有部分接口以便进行任务检查。笔者认为，对于子任务，一些核心的方法，设计思想如下：
在 SubTaskInfo 类内部实现的方法：

- 子任务的定位（多态）
  static SubTaskInfo *getnowTask(QVector<SubTaskInfo *> tasklist, QString &nowname);此函数重载了 TaskInfo 的 getnowTask 函数，传入的参数列表为 QVector<SubTaskInfo *>的子任务动态数组，以及子任务的唯一标识——子任务名称。
- 子任务的动态更新
  如上文所说，笔者采用了全局变量 CHANGE_SUB 存放数据变化的子任务，因而笔者同样重载了 TaskInfo 的 taskUPDATE 函数，进行子任务的数据更新。
  virtual void *taskUPDATE*(QVector<SubTaskInfo *> *subtasklist);

在 UserInfo 及其子类中实现的方法。
用户模块对于子任务的操作，主要是由 u_LEADER 与 u_TRAN 实现。
负责人模块，主要是分割原始任务生成子任务，查看子任务，发表反馈，汇总子任务。
译者模块，主要是查看子任务，进行子任务翻译，子任务反馈查看，子任务译者互校，提交子任务。
以下分模块阐述：

u_LEADER

- 负责人分割原始任务生成子任务
  主要通过方法
  void l_userDivideTask(TaskInfo *task, UserInfo *tran, QString &taskcon);
  实现。负责人在界面中对任务文本进行恰当分割后，确认完毕后调用此函数，相应的文本就分配给对应的译者。在界面的呈现上，笔者采用了动态数组 QTextEdit 与 taskTranslater 组合，界面动态生成所需译者数量的文本编辑区域，负责人只需要完成文本的分块。当然，在负责人分割任务之前，系统需要提示负责人当前是否已经确认译者名单，允许负责人考虑是否确认分割。
- 负责人查看子任务

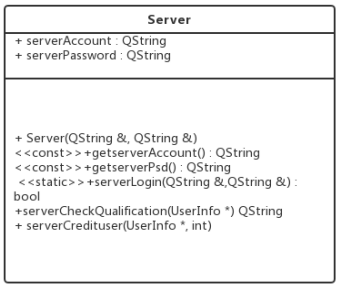
负责人具有查看所有子任务的接口。在查看时，系统需要预先判断目前任务是否已经分割，同时系统也可以提供一个判断方法用于判断译者是否已经提交译文。

- 负责人发表反馈
  负责人查看已经提交的译文时，可以对相应的译文提出反馈意见。反馈意见在图形界面的 QTextEdit 输入，系统首先需要判断输入内容是否为空。另外，负责人确定反馈后，应该具有查看历史反馈的接口，允许负责人查看自己与译者子任务的反馈意见，同时负责人也可以看到其他译者对本译文的反馈意见。
- 负责人汇总子任务
  负责人在确认提交任务之前，需要有进行子任务汇总、确认最终译文的接口。负责人的界面应该同时具有自主的内容编辑窗口 QTextEdit 与子任务的浏览窗口 QTextBrowser，负责人可以选取译者的译文进行编辑完成最终译文。
- 负责人提交任务
  负责人具有最终提交完整翻译任务的接口，此时译文内容将存储在原始任务的译文内容之中。因此，笔者在进行负责人类的设计时，有意让负责人的操作同时能够对于原始任务和子任务发生作用。

u_TRAN

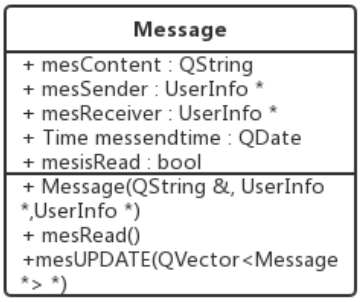
- 译者查看子任务
  译者具有查看自己子任务内容的接口，查看的内容应该包括子任务所分配的任务内容，子任务暂存的译文，子任务上一次提交的译文，以及子任务的其他译者信息（为了方便译者互校的设计）。
- 译者进行子任务的翻译
  译者进行子任务的翻译，实质上就是对子任务的 subTran，subTrantemp 的修改。译者翻译结束后，可以选择保存目前的翻译内容，同时也可以选择直接提交。在图形界面上直接体现为任务内容的文本浏览器和翻译内容的文本编辑框。
- 译者查看子任务反馈
  译者可以查看自己所进行的子任务的反馈情况。译者应该能看到反馈的具体内容、反馈的发出者等基本信息，因而笔者在 SubTaskInfo 中设计了 QVector<Comment *>的动态数组，单独设计 Comment 进行反馈的保存，使得反馈模块化。
- 译者提交子任务
  译者具有提交子任务的接口，此时提交的任务将会经过负责人的审核，同时若存在反馈，译者将会受到系统的消息提醒，译者需要针对反馈进行修改并再次提交。
- 译者互校
  译者互校是笔者设计的附加功能，其具体实现于后文附加功能中呈现。

### 3.1.3 平台类（Server）




平台 Server 类主要用于后台修改用户积分。这一部分也是附加功能，将会在后文重点阐述，这里只做简单分析。
其属性包括管理员账号与密码，用于登录识别。
QString serverAccount;
QString serverPassword;
对于平台，能够查看系统所有用户的基本信息，尤其是用户注册时填写的语言资质证明（userQualification），人工核验后可以设定用户初始积分。

### 3.1.4 消息类（Message）

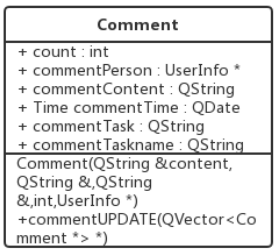



消息类包含的属性有：
QString mesContent;用于存放消息内容。
UserInfo *mesSender;用于存放发送消息的用户指针。担当此指针为空时，表示消息来自系统，换句话说，当消息由系统发出，则消息的 mesSender 为空地址。
UserInfo *mesReceiver;用于存放发送消息对象用户的指针。
QDateTime messendtime;用于存放发送消息时的系统时间，在存放数据时与发送者、接收者共同构成了消息的唯一标识。
bool mesisRead;用于标记消息是否已经被接收者阅读。消息最初构造时，其值

为 false，当消息接收者阅读消息后，将会调用相应的成员函数 mesRead，修改为 true。
以下阐述消息类方法的实现思路：

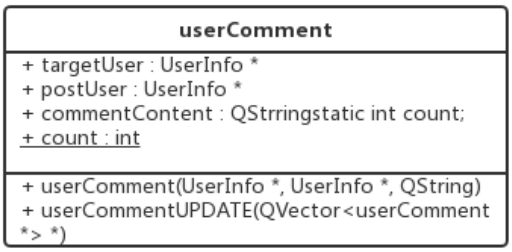
- 消息的构造
  在设计类时，笔者设计了一个含参的构造函数：
  Message(QString &content, UserInfo *receiver,UserInfo *sender)
  其功能是生成一条新的包含发送者、消息接收者以及消息内容的消息。
- 消息的更新
  笔者设计了全局动态数组 ALL_MESSAGES 与 CHANGE_MES 分别用于存放系统所有消息以及新增的消息，后者将在数据库存放阶段用户更新数据库。由于消息类的特殊性（即，消息已经发送即不可更改），所以 CHANGE_MES 所存放的就是本次程序运行所发送的全部消息。
- 消息的发送（多态）
  在程序设计中，笔者的程序实现了系统发送消息与用户发送消息的多态。这一部分涉及到附加功能：用户与用户的消息交流，故将于后文详细阐述。

### 3.1.5 反馈类（Comment）



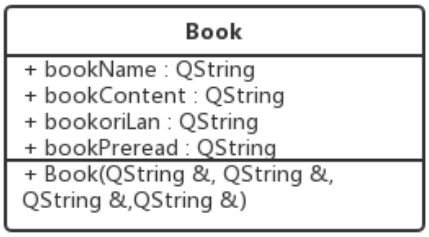

反馈类主要是讲负责人和译者的反馈整合成为一个模块。
其包含的属性有：
int count;用于记录反馈的序列号信息，并无实质内容，但在数据库存放时作为反馈的自增约束键。
UserInfo *commentPerson;用于存放反馈提出用户的指针。由于笔者的设计，这个用户可能是负责人，也可能是译者。
QString commentContent;用于存放反馈的具体内容。
QDateTime commentTime;用于记录反馈发出时的系统时间，译者查看反馈时将会看到反馈发送的时间。
QString commentTask;用于存放反馈所针对的译文内容，原本笔者的设计是为了便于译者查看历史反馈以及历史提交译文之间的关联，但后来发现并没有这种设计的必要，故这一属性的用途在程序中并没有体现。
QString commentTaskname;用于存放反馈所针对的子任务名称，也就是子任务的唯一标识。
对于反馈类，由于反馈主要是用于用户模块的反馈操作，在用户与任务模块已经提及，因此其具体方法在这里并不赘述。

### 3.1.6 用户评论类（userComment）




用户评论类是笔者单独设计的附加功能，其具体实现已经在后文附加功能中具体阐述，此处不再赘述。

### 3.1.7 书籍类（Book）




书籍类的设计，主要是为了将任务中的原始文本模块化，在程序中并未体现直接的方法，故此处只是介绍一下书籍类的属性。

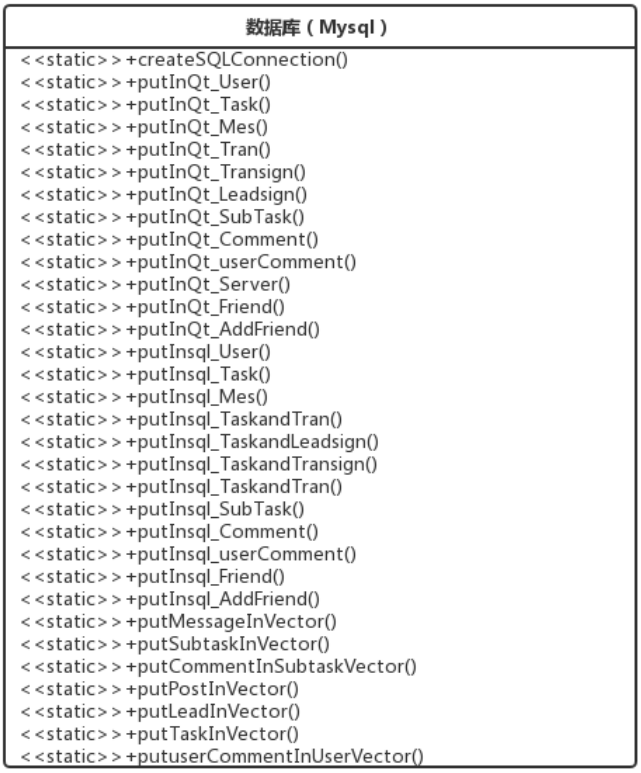
QString bookName;此属性用于存放书籍名称，由任务发布者手动编辑，并且书籍名称与发布者标识将会共同构成任务名称。

QString bookContent;此属性表示书籍内容，也就是待翻译的任务原文，在笔者的设计中，书籍内容通过 txt 文件上传实现。

QString bookoriLan;此属性用于存放任务的原始语言，也是通过任务发布者手动编辑。

QString bookPreread;

## 3.2 数据库/文件结构设计




数据库存在十二张表。

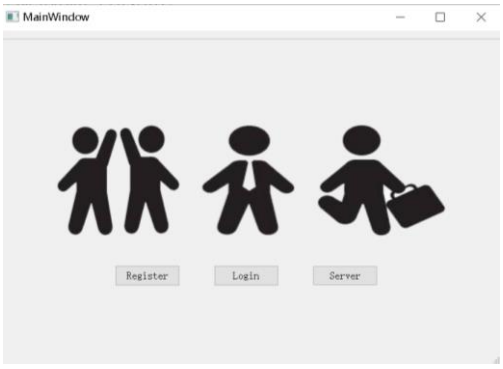
t_addfriend |存储发送好友申请的数据（中间表）
t_comment |存储子任务反馈数据
t_friend |存储好友关系数据（中间表）
t_message |存储消息盒子数据
t_server |存储平台管理员数据
t_subtask |存储子任务数据
t_task |存储任务数据
t_taskandleadsign |存储任务与负责人报名人数据（中间表）
t_taskandtran |存储任务与译者数据（中间表）
t_taskandtransign |存储任务与译者报名人数据（中间表）
t_user |存储用户数据
t_usercomment |存储用户评论数据

## 3.3 界面结构设计

### 3.3.1 界面结构

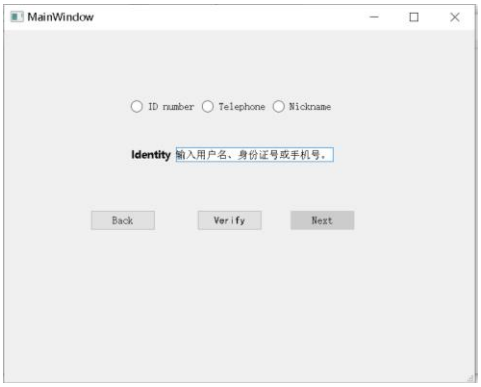
#### 3.3.1.1 主界面（MainWindow）

用户打开程序，首先看到主界面。
用户可以选择 Register 进行注册，选择 Login 登录，选择 Server 登录平台管理员进行后台操作。



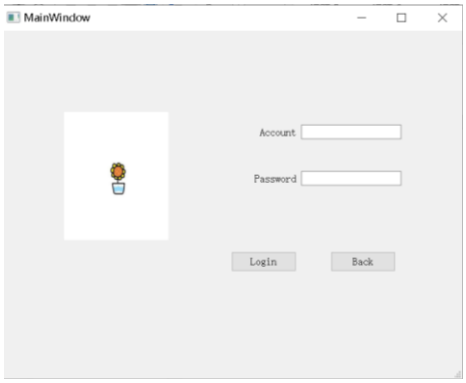

#### 3.3.1.2 注册界面（RegisterWindow）

Register 界面主要运用了 StackedWidget 容器。
用户选择 Register 注册界面后，首先需要输入合法的唯一标识。笔者设计了单选框允许用户选择自己的唯一标识类型（身份证号、手机号或者昵称）。
若标识合法，将利用 StackedWidget 容器跳转到用户密码确认界面和个人信息维护界面，用户需要在这两个界面中填写必要的个人信息，以及语言资质证明。



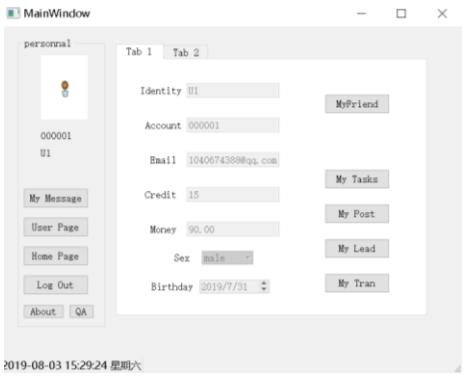
### 3.3.1.3 登录界面（LoginWindow）

用户登录界面，笔者设计了头像的插入，使得界面更美观。




### 3.3.1.4 个人界面（PersonMainWindow）

用户登陆成功后，将进入 PersonMainWindow 界面。此界面是本平台许多核心功能的界面，需要通过此界面跳转到其他界面。
首先，在本界面，用户可以选择个人主页（UserPage），在个人主页中用户可以浏览自己的个人基本信息，维护个人信息，进行充值操作。同时也可以查看我的好友（MyFriend）、我的各类任务（MyTasks、MyPost、MyLead、MyTran）。
用户可以选择进入平台主页（HomePage），在主页，笔者设计了最新任务和个性化推荐两部分内容。最新任务将会呈现发布时间最近的任务，个性化推荐则会呈现与用户之前的任务相类似（原始语言或者翻译目标语言相同的任务）。在主页，用户可以选择查看所有任务、正在招募的任务，也可以选择发布任务。同时，笔者也设计了搜索任务的功能。
用户也可以点击我的消息（MyMessage）查看我的消息，包括我收到的消息，我发送的消息，同时也可以编辑发送消息。
另外，用户还可以点击 About 查看关于我们的平台信息，点击 QA 查看常见问题与解答。



### 3.3.1.5 消息界面（MessageWindow）

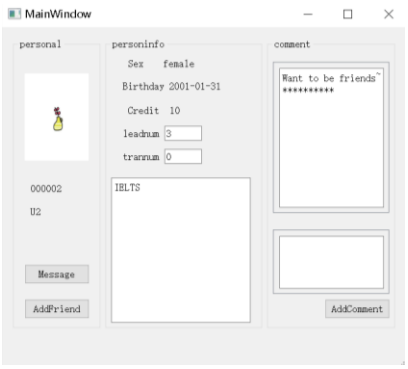
消息界面主要也是基于 StackedWidget。用户可以选择查看我收到的消息（MyReceive），可以选择查看我发送的消息（MySend），上述消息都会以 QTableView 的形式呈现，并呈现消息发送者（或接收者），消息是否已经阅读，消息发送时间，消息内容。用户可以点击查看，则界面进入单独消息查看界面。在单独的消息查看界面，用户可以看到完整的消息内容、消息发送者或者消息接收者的头像及昵称。若是查看收到的消息，用户也可以选择恢复（Reply）消息。

### 3.3.1.6 发布任务界面（PostTaskWindow）

由于笔者最初的设计思路不是很严密，所以发布任务变成了一个单独的窗口。在本窗口，用户可以进行任务基本内容的编辑，包括上传任务文本，撰写任务简介、确定任务酬金等等，同时用户可以为任务设定任务封面。当发布者输入有误或不符合格式时，界面会弹出错误提醒。

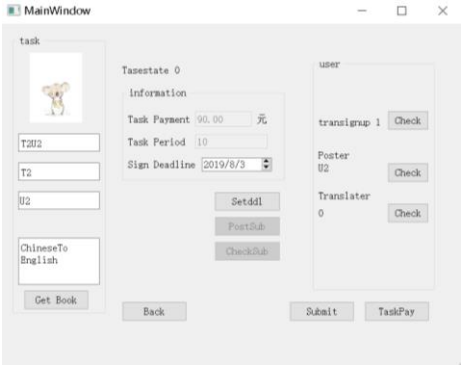
### 3.3.1.7 查看用户界面（CheckUserWindow）

用户存在多种方式查看其他用户：例如查看我的好友、查看任务发布者、查看任务报名人等等。下图即用户查看其他用户是时的界面。其中包括用户头像、用户昵称、用户性别等用户基本信息。
查看的用户可以看到该用户收到的评价（Comment），同时自己也可以补充评价（AddComment）。另外，笔者设计了发送消息（Message）与（AddFriend）的按钮选项。用户点击发送消息时界面将会跳转到消息界面，用户即可发送消息给指定的该用户。当用户点击添加好友按钮时，若双方已经成为好友，系统会对用户进行提示。否则，则会呈现成功发送好友请求的对话框。
当用户查看的用户是一个数组时（例如，查看任务译者），此时，用户查看界面将会利用 StackWidget 跳转到 QTableView 的页面，页面呈现用户的头像 icon、用户昵称以及账号，以及供查看的用户点击的查看（Check）按钮，点击后即进入详细的查看页面。

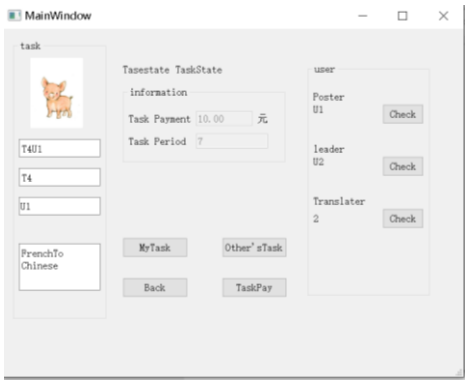


### 3.3.1.8 查看任务窗口（MytaskWindow）

查看任务窗口基于 StackedWidget 设计，包括了任务队列表单（QTableView），针对任务发布者、任务负责人、任务译者、与任务无关的人的不同查看窗口，并根据任务的具体状态，其部分控件稍有区别。
当用户点击我的任务或者点击选择正在招募的任务列表，查看任务窗口将会以 QTableView 的形式呈现符合条件的任务。其中包含任务名称、任务发布者、任务状态以及当前用户在该任务中的状态（Visitor、Poster、Leader、Translater）。用户可以点击查看（Check）按钮进一步查看任务信息。
对于发布者，其查看界面除了包含有任务的基本信息之外，还具有对任务负责人、任务译者、任务负责人报名者、任务译者报名者的查看权力，同时也具有获取任务全文的按钮（Getbook）。任务查看窗口存在任务状态的 label，会随着任务状态的改变而变化。另外，负责人也具有结束任务、确认支付酬金的按钮。
对于负责人，其查看界面除了必要的任务信息，还具有查看任务发布者、查看任务译者、查看任务译者报名人的按钮，同时也具有确定译者报名截止时间、分配子任务、查看子任务、上交任务、领取酬金等等按钮设置。（如下图即为人物负责人的查看窗口）

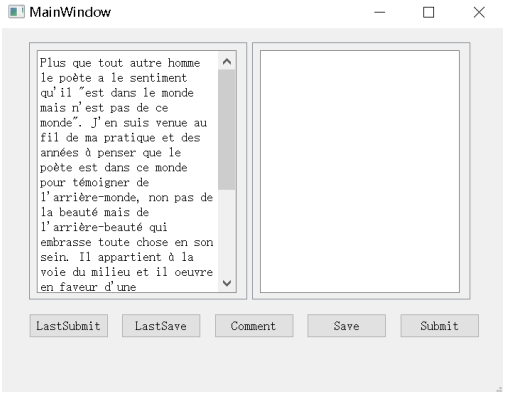



对于译者，其查看窗口对于原始任务的信息较少，基本上即查看任务发布者、查看任务发布者、查看任务负责人及译者等。具有较多操作子任务的按钮。如查看并编辑我的任务，查看其他译者的任务，领取酬金等等，点击领取酬金，系统会根据任务支付状态对译者进行提示，点击查看其他译者的任务，系统将通过 QTableView、以及任务编辑窗口，让用户跳转进入对其他译者任务的编辑窗口。点击查看我的翻译任务，同样跳转到任务编辑窗口。



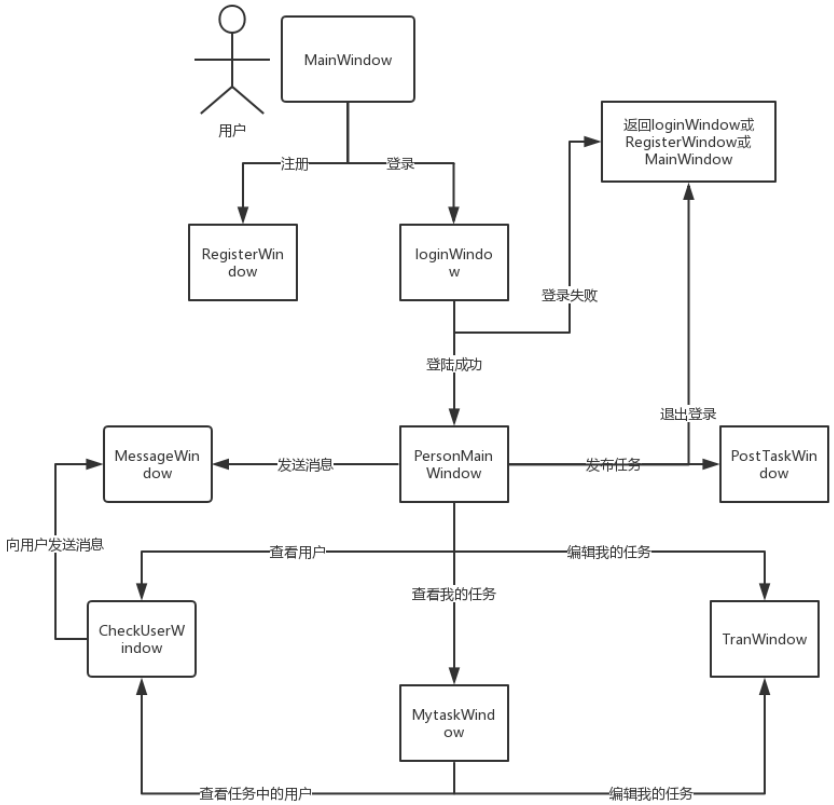
#### 3.3.1.9 任务编辑窗口（TranWindow）

任务编辑窗口的窗口名字可能会导致用户理解产生误差。实际上，发布者查看完整译文、负责人进行子任务的分割以及反馈意见、提交最终任务的编辑、以及译者的翻译任务都是在此窗口进行。对于不同的用户，其呈现结构是有差异的。
对于发布者，若选择查看原文（也即是 getbook），窗口将呈现任务原文。若选择查看已经完成的译文，窗口会呈现任务原文和译文。
对于负责人，当负责人进行子任务的分割时，窗口会以 Toolbox 的形式，按照译者的数量提供相应的文本编辑框供负责人进行编辑。当负责人进行子任务的查看与反馈时，窗口则会呈现负责人所选择的子任务原文以及译文，并且提供了供负责人点击选择的反馈（Comment）、查看译者（Translater）功能按钮。当负责人提交任务时，窗口存在保存（Save）、刷新（Refresh）和确认提交（Confirm）按钮。
对于译者，主要是对自己任务的编辑窗口，并且提供了查看上一次提交、查看上一次保存、查看反馈、保存、提交等按钮。译者可以对比原文进行任务的翻译。当译者选择查看其他译者的任务时，窗口则会呈现译者所选择的译者的任务窗口，其中包括该译者的任务原文、已经提交的任务译文。译者具有反馈（Comment）的选项按钮对该译者的任务进行点评，实现译者互校。




## 3.3.2 界面跳转关系

界面的主要跳转关系，可以用以下流程图简单描述。




# 3.4 关键设计思路

## 3.4.1 容错设计

本平台的容错面向的包括：用户输入错误、用户状态不符、用户重复操作等。

### 3.4.1.1 用户输入错误

- 日期设计的错误，例如用户出生日期的编辑、任务报名截止日期的编辑，平台会在内部进行检验，然后在界面通过对话框进行提示。核心标准是，用户出生日期不得晚于当前系统日期，用户设定的报名截止时间不得早于系统当前时间。
- 用户注册身份证、手机输入错误。用户注册时，函数会进行检查。例如，若用户选择用手机号完成注册，则系统会检查用户的输入是否为 11 位数，是否包含字母，并给出相应的提示。
- 用户账号密码错误。用户的账号密码错误存在以下情况：账号或密码为空、账号不存在、密码错误等。对于每一种情况，平台都有相应的应对函数和对话框提示，并要求用户重新输入直到正确。
- 用户密码确认。用户在进行密码确认时，系统会检查密码输入框是否为空、密码与再次

确认的密码是否一致，若不一致，将会清空密码输入框，对话框要求用户重新输入。

- 用户没有上传原文任务。

#### 3.4.1.2 用户状态不符合条件

- 当用户申请成为任务发布者时，需要检查用户积分是否达到发布者的要求。
- 当用户申请报名参与任务时，需要对用户是否已经参与任务、是否已经报名进行检查。

#### 3.4.1.3 任务状态不符合条件

- 当任务发布者未确认支付酬金时，负责人与译者无法领取酬金。
- 当任务报名截止时间没有确定是，截止时间默认为负责人报名截止时间，并且此时认为负责人还没有进行招募，故不允许直接分配子任务。
- 当负责人还没有确定译者名单时，不允许对任务进行分割，提示负责人还没确定译者。
- 当负责人已经分配任务后，即不可以再次分配任务

### 3.4.2 运行过程描述

程序主要的运行过程为：首先，程序启动，数据库与程序搭建连接，数据库外部数据存入内存。此后用户进行相应的操作，操作内容将会在内存中进行，并且在内存中保存对数据的更改。退出程序后，程序将更改的内容存入数据库。

## 3.5 附加功能实现

### 3.5.1 用户发送消息功能（多态）

在笔者的设计中，对于用户类设计了一个返回值为空的
`void UserInfo::mesSend(QString &content,UserInfo *receiver, UserInfo *sender)`，并进行了重载
`void UserInfo::mesSend(QString &content, UserInfo *receiver, UserInfo *sender)`
二者参数列表不同，所实现的功能均是发送消息，并且都由用户调用。区别在于，前者是用户发送给其他用户消息时的实现方法，而后者是系统向用户发送消息或提醒时调用的方法。参数列表中，具有三个参数的函数三个参数分别表示消息内容，消息接收者，消息发送者。具有两个参数的函数的参数分别表示消息内容和消息接收者。
在实际的设计中，用户可以通过图形界面，用以下三种方式进入发送消息编辑窗口并发送消息。

- 用户可以通过个人界面点击“SendMessage”发送消息给其他用户，此时系统会对用户是否存在进行验证。

- 用户也可以在查看我收到的消息“MyReceive”时点击回复消息“Reply”，这时函数会自动将回复的对象作为接收者。
- 用户可以在查看其他用户的个人信息页面时点击“SendMessage”向该用户发送消息，此时函数会自动将用户所查看的用户所谓消息接收者。

Mssage 类的主要属性成员包括：
QString mesContent;用于存储消息内容
UserInfo *mesSender;表示消息的发送者。若为空，则表示此消息由系统发送
UserInfo *mesReceiver;表示消息的接收者。
QDateTime messendtime;用于标记消息发送时间。
bool mesisRead;用于标记消息是否阅读。消息构造时，默认值为 false，当用户阅读消息时，会调用阅读消息函数修改值为 true。
主要的方法：
Message 类内部方法，或者说本模块自带的方法：
void mesRead();用户阅读消息时调用，调用后消息的阅读状态 mesisRead 会从 false 改成 true。
void mesUPDATE(QVector<Message *> *meslist);用于更新数组中的消息数据。在本程序中，主要体现为对于全局变量 CHANGE_MES（存放新增的或者信息发生变动的消息）的更新。
UserInfo 类的方法，即用户对于消息的操作：
void mesSend(QString &content,UserInfo *receiver, UserInfo *sender);本函数用于用户向其他用户发送消息。
void mesSend(QString &content,UserInfo *receiver);本函数用于系统向用户发送消息，重载了用户向其他用户发送消息的函数参数列表，与上述函数构成多态。
int mesunRead();本函数用于记录用户未读消息总数，并返回其值。通过调用此方法，用户会在登录时得知自己消息盒子的变动（收到多少新消息），从而完善消息盒子的提醒功能。

### 3.5.2 用户评价功能

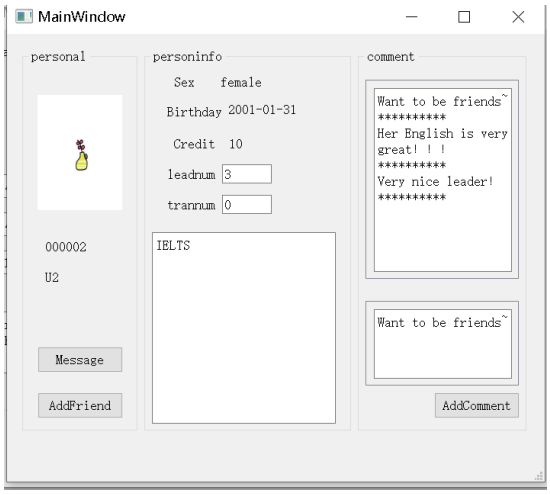
在笔者的设计中，对于用户增加了用户评价模块，旨在让用户互相之间发表对于对方的看法。在类的设计中，体现为 UserInfo 类中 userComment 的动态数组。笔者设计的 userComment 类包含以下内容：

静态成员 static int count;用于标记评论的数量，另一方面则是作为评论的识别标志。
UserInfo *targetUser;表示用户评论的对象。
UserInfo *postUser;发表发表评论的用户。
QString commentContent;存储评论内容。

在数据库的存储中，以 count 作为唯一自增键约束其他内容。
当用户查看其他用户个人信息时，可以在 comment 浏览区间（文本浏览区）查看到用户的历史评论，同时，用户也可以在评论区下方的文本编辑框编辑对于

该用户的评论，评论区对于用户的评论会实现实时更新，同时评论区将只呈现评论的内容而不呈现评论发布者和评论时间。

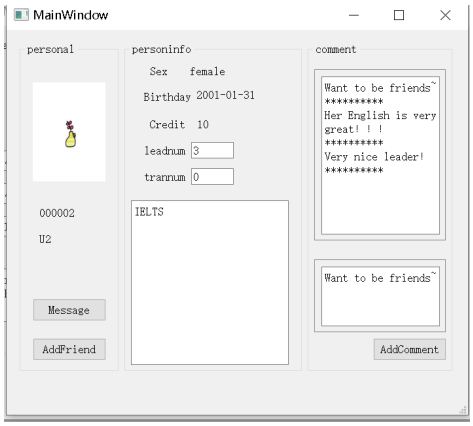
实现的功能函数为：
void adduserComments(UserInfo *target,QString &comment);
该函数由发送评论的用户进行调用，参数列表包含评论的对象用户和评论的内容，函数体根据传入的实参评论对象、评论内容以及调用此函数的对象和系统时间构造评论。

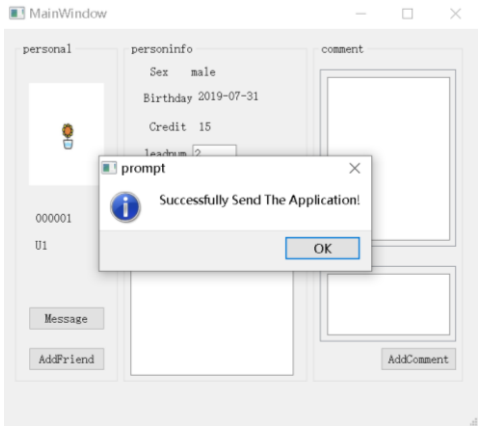


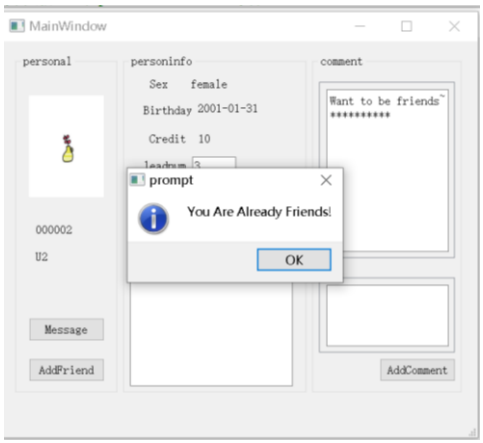

### 3.5.3 用户好友功能

在笔者的设计中，用户与用户之间存在好友的设计。
在查看用户界面，如下图所示，用户界面左下角存在“AddFriend”选项按钮。用户点选后，首先会检查用户与目标用户是否已经是好友关系，若双方已经成为好友，系统将给出提醒。若双方并不是好友，则系统会向目标用户发送本用户的好友申请。在数据的储存上，则体现为发送好友请求的用户，其存储已申请添加好友的动态数组
QVector<UserInfo *> userSendfriend
会动态增加目标用户的指针。
而目标用户的待同意好友申请动态数组
QVector<UserInfo *> userRecfriend
则会动态增加发送好友申请的用户指针。
好友申请的判断与数据更新
bool applyuserFriend(UserInfo *targetuser);

实现。其返回值为 bool 类型变量。若为 true，则成功发送好友申请，否则说明双方已经成为好友，系统会给出相应提示。（下图中，）方已经成为好友，系统会给出相应提示。（下图中，从上至下分别呈现用户申请好友的按钮，用户成功发送好友申请，用户双方已经成为好友，申请失败的界面情况）。




### 3.5.4 译者互校功能

在笔者的设计中，同一任务的不同译者之间可以通过内容反馈的形式实现译者互校。

对于一个已经分隔好的任务，译者 A 完成属于他的子任务 A 并提交任务后，

首先是可以接收到来自负责人的反馈。同时，该任务的其他译者，如译者 B，可以通过查看其他译者的任务查看到 A 所提交的原文和译文文本。B 对于 A 的译文不具有直接修改的权限，但是可以通过意见反馈（Comment 选项）提出修改意见。对于译者 B，可以查看自己的发表的历史评论，但其他译者所做出的反馈，以及任务负责人所做出的的反馈，译者 B 则不具有查看权限。

译者与负责人的评论都存储在 SubTaskInfo 类中的动态数组：

QVector<Comment *> taskFeedback

Comment 类的属性如下：

int count;用于记录评论在整个数组中的序列。在数据库储存时也是作为自增的约束键。

UserInfo *commentPerson;用于存放提交反馈的用户，例如某任务的负责人指针。

QString commentContent;用于存放反馈内容。

QDateTime commentTime;用于记录反馈的时间。

QString commentTask;用于记录所反馈的任务的译文内容。

QString commentTaskname;用于记录所反馈的子任务名称。

Comment 类内部所实现的方法有：

void commentUPDATE(QVector<Comment *> *commentlist);主要用于更新全局变量 CHANGE_COMMENTS，存放新增的或发生改变的任务反馈，以便数据库存储。

在用户模块中所实现的方法：

对于负责人 u_LEADER：

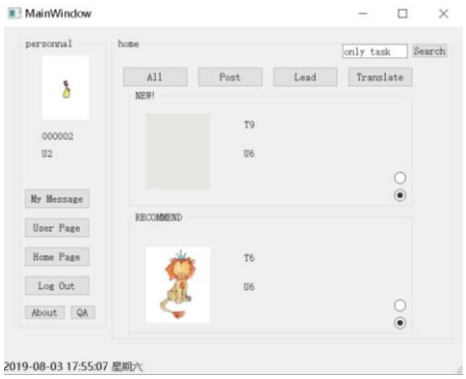
void l_userFeedbacktask(SubTaskInfo *subtask, QString &feedback);其中参数列表分别为待反馈的子任务以及反馈的内容，函数体会通过传入的参数以及调用函数的对象构造 Comment；

对于译者：

bool t_usercheckComment(SubTaskInfo *task);译者可以查看自己子任务的反馈。若不存在反馈，则返回 bool 值为 false。

void t_userCommentother(SubTaskInfo *task,QString &comment);译者可以以译者身份向同一任务下的其他译者译文进行反馈，函数的实现与负责人的反馈函数基本相似。

### 3.5.5 任务个性化推荐



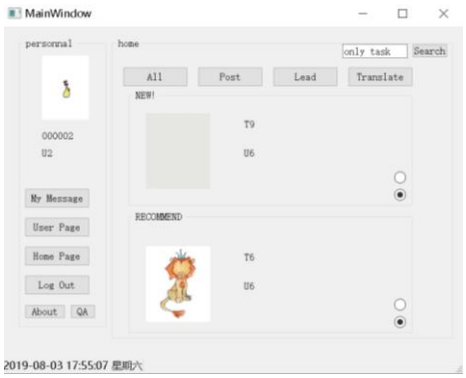

笔者设计了任务个性化推荐界面。其算法思路为：每一个用户（UserInfo）存在一个寻找兴趣任务的方法（userFindInterest），通过此方法，在系统所有任务中寻找用户不参与并且其文本原始语言与任务目标语言与用户的某个任务相同的任务，作为语言相同的任务进行推荐。

实现方法如下：

QVector<TaskInfo *> UserInfo::userFindInterest()，函数首先遍历系统所有任务，寻找出用户不参与的任务。然后再在这些任务之中寻找任务原本语言或者任务目标语言，与译者所参与的任务的原始语言或目标语言相同的任务，并向其放入一个动态数组中并返回。

用户可以在主页（HomePage）查看推荐任务，推荐任务最多有两页，通过单选框勾选查看，界面会呈现该任务的书籍名称和发布者昵称。

### 3.5.6 最新任务面板




笔者设计了最新任务浏览面板。该界面会呈现当前最新发布的任务信息（任务封面、任务书籍、任务发布者），从而呈现当前最新发布的任务。

该功能的实现方法为：在从数据库向内存中存储数据是，采用（ORDER BY）的方法，使得数据依照任务的发布时间正序排布，同时存在新任务时，其插入动态数组的方式也是与数据库的插入方式相吻合的。因而最新任务可以直接抽取数组中最后的任务（也就是最新发布的任务）

需要注意的细节是，当任务个数小于 1、小于 2 时，需要有相应的条件语句对界面的排版进行改变，以免造成数组越界。

### 3.5.7 平台管理员

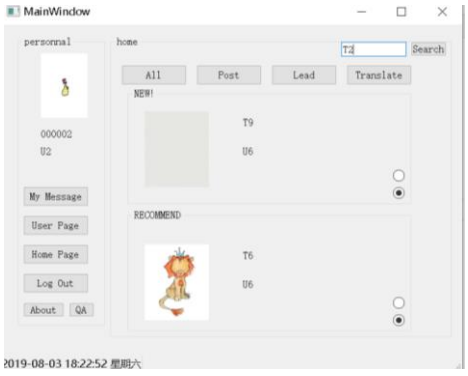
笔者设计了平台管理员的功能，允许用户以管理员的模式登录系统，修改用户积分。这一部分的设计相对来说比较简单，笔者仅仅是为平台管理员（Server）设计了账号密码属性，以及获取用户语言资质证明、设定用户积分的接口。在进入主界面时，存在一个 Server 按钮，用户点选后，凭借正确的账号密码即可进入管理员模式，进入系统后，用户将会查看到所有用户的数据。点击查看某一个用户后，即进入该用户的个人界面窗口。通过审查该用户的语言资质证明，管理员可以直接修改用户的积分数值。

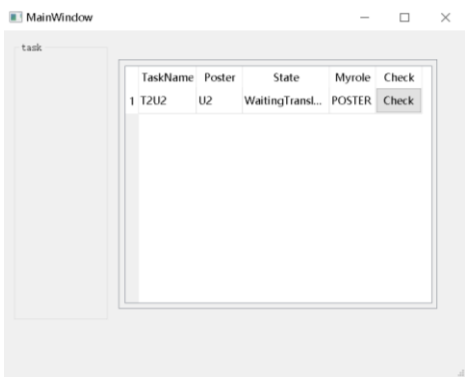
### 3.5.8 多语种互译

在平台设计中，笔者是通过 Combobox 进行翻译语言选择的，所以理论上任意语言都能够进行双向翻译。但在本次展示中笔者只呈现中、英、法三语的翻译。任务的原始语言和目标语言会在发布者进行任务信息完善时，发布者通过选择 combobox 中的选项，然后系统直接读取文本内容作为相应的语言存储在任务的属性之中。

### 3.5.9 搜索功能

笔者实现了平台的搜索功能。在用户登陆后的首页（HomePage），用户可以在右上角输入欲搜索的任务名称进行搜索。搜索将返回列表，用户可以点击相应的查看按钮继续查看。



# 4 项目总结

## 4.1 开发难点

最初开始进行设计时，我有点一筹莫展，甚至连划分哪些类、平台实现哪些功能，都是只有一个模模糊糊的感觉。后来随着自己在网上查阅一些开发资料，以及对于 QT 编程界面的逐渐熟悉，包括老师的提醒“大作业要早点开始准备”，我很快就开始了用户类和任务类的大致划分，以及整个平台翻译任务完成的大致流程设计。

在这之中，我认为比较难的，在于用户与用户的继承关系，用户与任务的交互。在最开始，我一直困惑的地方在于，当任务中的某一个角色的某个操作导致任务数据发生改变时，这一改变如何让同一任务的其他角色也能够获取？后来我是通过设计全局变量，以及用户中包含任务指针数组，任务中也包含用户指针数组，这样的双向定位方式，实现了既能够从用户端定位任务，同时也能从任务端定位用户。

第二个难点，则是大量指针变量的使用，或者说，这不是难点，而是很容易因为编程能不规范而造成程序崩溃的易错点。在本次平台设计中，存在大量的指针变量，稍不注意没有初始化，程序便异常终止。在这一次大作业的完成过程中，笔者就有不少次因为指针变量的问题而一行代码一行代码进行 qDebug。但与此同时，这也在一定程度上提高了笔者变成时思想的严密性。例如，在进行用户指针检索时，笔者会首先判断该指针是否为空，然后再进一步进行操作，这对于笔者养成更好的编程习惯是很有帮助的。

第三个难点，在于图形界面编程。笔者很早就开始采用 QT 进行平台设计，但在进行界面布局以及界面与界面之间的跳转时依然遇到了不少问题。比如说不同窗口之间的信号与槽机制、窗口与窗口之间共享数据的分享，等等，这些都需要在短短三周时间内理解并掌握。对于笔者，尽管这一部分所花的时间并不是特别长，但依然特别“烧脑”。

第四个难点，则是数据库存储数据机制。在本次程序设计中，笔者采用的存储模式是数据库 Mysql，一开始由于 Qt 位数问题，数据库无法加载，后来在一次又一次的安装与卸载之后终于成功实现了数据库与内存的读取。而另一方面，由于对于数据库的操作是通过 query 语句实现，因此 Qt 不会对该语句进行语法检查，所以需要编程者自己保证语法正确，才能进行数据库与内存的数据读取操作。

## 4.2 遇到的问题以及一些解决方法

在这一次开发过程中，笔者遇到了不少之前程设课程没有也不会遇到的问题。一方面，这些可能是由于编程习惯不佳、也可能是因为不熟悉操作而导致的问题，确实常常需要笔者用一两天的时间寻找错误，另一方面也让我在一些细节上，以及解决具体问题的思路上有了一些启发。

第一，是遇到问题多查多问。我记得课程上老师和助教都强调过，不要以为你在

编程过程中遇到的问题只有你遇到过。上网查询程序的相关报错，确实能够让自己很快找到错误的原因。在本次编程过程中，笔者就利用 CSDN、博客园、Stack Overflow 中找到了自己需要的不少问题的答案。
第二，是经常备份。当代码量较大时，养成保存备份的好习惯，才不至于当程序出现大面积瘫痪时不知道错误在何处。
第三，是先设计，思路清晰后再完善代码。笔者本次的设计，相对来说还是比较“丑”的。一方面是类的封装性没有很好的体现；另一方面，则是有一些应该属于同一模块的方法由于一开始设计的不严谨暴露在外，导致程序并没有能很好的实现 opp 的编程思想。我想这和自己面向过程的传统编程思路有关，另一方面，也是因为自己在设计之初并没有设计好整一个平台不同模块以及界面之间的联系，导致了这种差错。

# 5 相关问题的说明

如果需要的话，请在此部分说明程序开发环境和执行环境的搭建方法及操作实例等相关问题。
（此部分内容可以为空）