毕业论文

程序设计基本类图

\

Qml

main.cpp

**Read\_excel\_to\_db.h**

**Read\_excel\_to\_db.cpp**

**Tool\_class.h**

**Tool\_class.cpp**

**Tool\_InterAction.h**

**Tool\_InterAction.cpp**

**程序流程**

QML为Qt传统的widget桌面软件开发程序的延申，吸收了Web化开发界面效率高，可维护性好等诸多优点，同时保持了它以C++为基础的高效率，语法很类似于Web的程序。

具体实现方式的以C++为载体，在C++的main函数里执行事件消息循环，维持一个后缀为.qml的页面用于与用户交互，同时自定义的类也是在main函数里与.qml交互。

Qml本身具有非常多的内置Javascript函数用于直接的进行某种操作，如访问数据库，较简单的功能均可用Javascript函数来完成，但有时需要高效率的时候考研使用编写C++的工具类以用于完成任务。

**阶段性任务**

1. 写好基本的程序框架，从Qml页面到main函数在到具体的工具类来完成整个程序功能。

**类的设计与实现**

1.Tool\_InterAction(前后端交互工具类)：由于Qml本身的设计是用Web页面来组织页面，用C++来负责业务逻辑，因此前端的信号要让C++拿到需要通过Qt特有的机制来实现。此类的设计有两个成员变量：

* 1. signal\_type （信号类型）
  2. signal\_info （信号内容）
  3. tool\_class （工具类）

本可以在此就实现所有的Tool\_Class的全部功能，但考虑到一个类的设计过于臃肿，且不便维护，便采用分层设计的思想，这一个类只是用来接收信号类型以及信号内容，判断信号类型以供选择具体调用什么Tool\_function();

1. Tool\_Class(工具函数类)：在此部分完成了大部分需要用C++完成的功能，其包含众多成员函数，受到上层Tool\_InterAction（）的选择调用
2. Word类，做为各个操作的基本成员变量，封装了作为一个单词应该有的属性以及get（）和set（）成员函数。

**主要功能的实现**

1. 从文件系统选择Excel写入到数据库。
2. 统计数据
   1. 词汇量
   2. 学习记录折线图
   3. 周期统计饼图
3. 学习方式
   1. 单词表顺序/乱序刷词
   2. 看词说意/看意说词
   3. 自动过词（后续可完善语音系统）
   4. 随机抽词
4. 单词本
   1. 本程序自带
   2. 用户自己导入
   3. 背单词过程中标记的单词
   4. 网络同步（后续待完善）

**页面设计**

初步设计是使用四个页面来承担不同的模块，分别为：

1. Page\_1 主要展示页面
2. Page\_2 单词本页面
3. Page\_3 图表统计页面
4. Page\_4 设置页面
5. Page\_1

Page\_1为主要页面区，是该软件的核心区域，主要承担的功能模块为刷词，具体可为：

1. 顺序学习方式展示单词（默认）
2. 乱序学习方向展示单词（可切换）
3. 随机抽词（可切换）
4. 自动过词（可切换）

页面的具体设计上，初步设计采用四个核心控制按钮的方式来操作单词的显示，分别为：prior（前一个），mark（标记），kill（秒杀），next（后一个）。但是这么设计的问题就是界面仍然是工控风，后期可以美化界面（通过扁平化设计，发光效果，阴影，图案等）

**数据库的设计**

1. 本程序考虑到实用性，需要在移动端进行，结合的具体情况，因此选择使用sqlite作为数据库，主要有三方面的内容：
   1. Word表
   2. 日志
   3. 辅助统计数据
2. 从Excel表写入Sqlite数据库使用C++工具类，用Qt提供的QAxObject类来操作。
3. 从Sqlite数据库读取到Qml页面使用集成的Javascript函数来完成。

**数据库初步设计：**

1. Word表，要存储的字段包括：
   1. word\_index(单词序号)： 用于单词管理，初步构思是为每一个单词添加一个独一无二的序号，但由于有不同的单词本，以及后续可以添加新的单词，这个功能可以更改为按照单词本来划分，而具体的操作流程为：在每一个新建的单词表（单词本，数据表）上来添加index，而这个index数量可以使用sql语句来统计数据，在调用具体的JavaScript函数添加新单词到具体的单词表里时候，再将index数据传入，辅助数据可以在数据库创建的时候同时添加：
      1. word\_list\_order\_word\_number(单词表单词数)
      2. word\_list\_order\_word\_current\_study\_index（当前学习序号）
      3. word\_list\_order\_word\_studied\_number(已学习的单词数)
      4. word\_list\_order\_studied\_number(该单词表学习遍数)

SQL语句:sqlite> select count(\*) from word\_list\_order;

* 1. word(单词字面)： 基础功能
  2. soundmark（音标）： 基础功能
  3. meaning（意思）： 基础功能
  4. study\_count（学习次数）:用于统计数据
  5. is\_marked（是否标记）：用于统计数据

SQL语句如下：

CREATE TABLE word\_list\_order( word\_index INT,word VARCHAR,soundmark VARCHAR,meaning VARCHAR,study\_count INT,is\_marked bool);

1. 辅助表，存储的字段包括
   1. 已经有的数据表表名。其一，可以利用触发器式的设计在建立新表的时候将新建的表名写入该辅助表 ，其二，可以利用在新建表的时候同时执行额外的sql语句在辅助表中写入数据。
   2. 存储的数据包括：
      * 1. 当前数据表的存储了多少条数据。

db\_table\_number;

* + - 1. 当前数据表学习到了那条数据。

db\_table\_current\_index;

* + - 1. 当前数据表已经学习了多少遍。

db\_table\_studied\_count;

* + - 1. 表名。

SQL语句：CREATE TABLE para\_info(

db\_table\_name VARCHAR,

db\_table\_number INT,

db\_table\_current\_index INT,

db\_table\_studied\_count INT);

1. 存在的问题，可能是由于JavaScript的问题，也可能是由于QML设计的问题，导致原先设计的要把一些控制变量以程序的变量的方式存储的方式，比如在程序打开的的时候默认选择第一个单词本进行学习，后来可以通过第四个页面的按钮进行切换的设计是不行的。我猜想的原因是各个QML页面各自加载自己的JavaScript文件，他们之间的JavaScript文件是没有什么关联的，而我欲用JavaScript文件来当作通信的中间物肯定是行不通的。所有需要再额外设置一个辅助表，用来存储一些基本的配置信息。
2. 额外数据表设计

辅助表extra\_info

SQL语法：CREATE TABLE extra\_info (

key VARCHAR，

value VARCHAR);

字段含义：key-value键值对来记录若干辅助信息。

程序基本流程：当新的数据表建立好以后，程序内部的有qml控制的JavaScript程序便只需要将原先存储在JavaScript文件中的变量，如var operate\_db\_table\_name;存储在数据库内，在由操作数据库的JavaScript函数来控制。get\_operate\_db\_table\_name()和set\_operate\_db\_table\_name()来获得。

SQL语句：INSERT INTO extra\_info values(

“operate\_db\_table\_name”,”word\_list\_order”);

问题的产生解决及心的：

1. 数据库的存储逻辑可能在一些情况下不好设计。例如单词表可以存储很多的单词，但是如果想要一些辅助数据，如单词本当前正在学习的index，那么在程序里面的时候可以设置一个变量标记即可。但是如何用数据库存储？为每个单词都添加一个字段标记？这样显然不行，因为如果存储了一万个单词便会存储一万次重复的标记，而且让这样一个全局的控制变量和局部的单词绑定在一个设计就是右问题的！对于这种问题，可以另外设计一个辅助表，其中可以专门设置字段来存储辅助数据。随之而来的问题是：专门建立一个数据表来存储很少的额控制变量有必要？实际上，可以把辅助数据库的字段设计的很抽象，让其能够满足很多的控制逻辑而不会用的很奇怪。比方说，标记学习到哪里的index不必要设计为index，这样相当于一个变量一个数据库，可以设计为key-value对，很多问题都可以基于此存储。

二．

**数据模型与程序内单词本的对应关系：**

1. 简略描述
2. 初步设计，一张数据库内的单词表会对应着程序页面内的一张单词本，但单词本数量用户可以自定义，比如用户点击收藏会将不会的单词写入自定义的单词本。
3. 基本的程序流程如下：

用户先在单词本页面（Page\_2）点击Add New新建一个单词本（填写信息包括单词本名称，标记日期，以及其他）。创建好一个单词本以后便可以将不会的单词收藏到该单词本内，以便后续集中复习。也可以点击comboBox选择Import,选择对应的单词本和本地文件上的excel，csv文件进行批量导入。

1. 也就是说每一个程序内的单词本只是数据库的一张表的映射，它实际上的数据模型里是没有数据的。而这个映射也仅仅是开发者对其进行的逻辑上的等价操作，使其同步，他们之间还是没有什么关联。
2. 创建单词本的时候填写的单词本名便成为了创建数据库表的表名。
3. 切换学习的单词本仅仅需要改变JavaScript函数

二． 细节描述

1. 在点击“Add New”后，弹出的页面上填写单词本名，并且在点击OK时触发响应函数onAccept（）首先进行写入一个新的单词本项进入listview的数据模型内，即刷新在程序上，而后续需要发出一个信号给JavaScript函数来创建一个对应的数据表（注：无需交付给C++处理）

2. 删除功能Delete同理

3. 导入功能的细节描述：点击展开ComboBox，选中Import，在文件列表里选择要导入的excel csv文件，点击确定后，将单词表名以及excel csv文件的路径传给C++，C++有了这两个参数便可以将单词数据读入对应的单词表。

4. 切换数据表进行学习的处理：由于此方法的敏感性，被放置到Page\_4页面进行操作。在用户点击Switch按钮以后，展示当前有的单词本，选择后改变JavaScript函数的参数，将将要执行的sql语句str以拼串的方式进行组装，便可达到改变数据表的目的。

**程序核心源代码：**

1. database.js

**set\_current\_DB\_index（）**

涉及操作： 前端页面传入的idx，在此写入数据库para\_info对应的数据表中

*//* *set* *current\_db\_index*

*function* *set\_current\_DB\_index*( idx )

{

*var* *db* = *getdatabase*();

*var* *result\_set*;

*var* *str* = "UPDATE para\_info SET db\_table\_current\_index="+*idx*+

" WHERE db\_table\_name='"+*operate\_db\_name*+"'"+

";"

*console*.log( *str* );

*db*.transaction( *function*(tx) {

*var* *result* = *tx*.executeSql( *str* );

*result\_set* = *result*;

*console*.log( *result*.rows.length )

*if* (*result*.rows.length != 0 ) {

*console*.log( " exec success ");

}

*else* {

*result\_set* = "Unknown";

*console*.log( " exec faliure");

}

})

}

readData\_by\_index()

涉及操作：在数据库里读取index对应的数据，在将结果拼列表的形式传给前端页面

*function* *readData\_by\_index*( idx ) {

*//debug*

*//* *console.log("readDate\_by\_index,* *The* *index* *is* *:* *"* *+* *idx);*

*var* *db* = *getdatabase*();

*var* *res*=""

*var* *res\_index*="";

*var* *res\_word*="";

*var* *res\_soundmark*="";

*var* *res\_meaning*="";

*var* *result\_set*;

*var* *sql\_str* = "select \* from "+*operate\_db\_name*+" where word\_index="+*idx*+";"

*console*.log( *sql\_str* );

*db*.transaction( *function*(tx) {

*var* *result* = *tx*.executeSql(*sql\_str*);

*if* (*result*.rows.length > 0) {

*res*=*result*

*res\_index* = *result*.rows.item(0).word\_index

*res\_word* = *result*.rows.item(0).word

*res\_soundmark* = *result*.rows.item(0).soundmark

*res\_meaning* = *result*.rows.item(0).meaning

*result\_set* = [*res\_index*, *res\_word*, *res\_soundmark*, *res\_meaning*];

*console*.log(*result\_set*[2])

} *else* {

*result\_set* = "Unknown";

}

})

*return* *result\_set*;

}

Load\_db\_table\_as\_word\_list()

涉及操作：程序初始话的时候将数据库里的数据表拿到，在传给前端

List\_view，将数据解析拼成JSON数据后append到数据模型处。

*//* *page\_2* *load* *database* *table* *as* *word\_list*

*function* *load\_db\_table\_as\_word\_list*( )

{

*var* *db* = *getdatabase*();

*var* *result\_set* = [];

*//* *var* *str* *=* *".tables;"*

*var* *str* = "SELECT name FROM sqlite\_master WHERE type='table' ORDER BY name;";

*db*.transaction( *function*(tx) {

*var* *result* = *tx*.executeSql( *str* );

*console*.log( *result*.rows.length )

*if* (*result*.rows.length > 0 ) {

*console*.log( "js: exec success ");

*for*( *var* *i* = 0; *i* < *result*.rows.length; *i*++ )

{

*var* *result\_temp* = *result*.rows[*i*].name

*console*.log( *result\_temp* );

*//* *result\_set* *=* *[]*

*result\_set*[*i*] = *result\_temp*

}

*for*( *var* *ii* = 0; *ii* < *result\_set*.length; *ii*++ )

{

*console*.log( "js: " + *result\_set*[*ii*] )

}

*//* *return* *";*

}

*else* {

*result\_set* = "Unknown";

*console*.log( "js: exec faliure");

*return* *result\_set*;

}

})

*return* *result\_set*;

}

Create\_db\_table() 前端页面创建单词本的时候创建一个数据表

*//* *create* *a* *db* *table* *at* *the* *same* *time* *the* *model* *add* *one* *new* *item*

*function* *create\_db\_table*( add\_table\_name )

{

*var* *db* = *getdatabase*();

*var* *result\_set*;

*var* *str* = "CREATE TABLE " + *add\_table\_name* +"( " +

"word\_index INT," +

"word VARCHAR," +

"soundmark VARCHAR," +

"meaning VARCHAR," +

"study\_count INT," +

"is\_marked bool);";

*console*.log( *str* );

*db*.transaction( *function*(tx) {

*var* *result* = *tx*.executeSql( *str* );

*//* *console.log(* *result.rows.length* *)*

*if* (*result*.rows.length === 0 ) {

*console*.log( " exec success ");

}

*else* {

*result\_set* = "Unknown";

*console*.log( " exec faliure ");

}

})

*//* *write* *para\_info* *to* *para\_info\_table*

*var* *para\_info\_str* = "INSERT INTO para\_info values('" +

*add\_table\_name* +

"', 0, 0, 0);"

*console*.log( "js: " + *para\_info\_str* );

*db*.transaction( *function*(tx) {

*var* *result* = *tx*.executeSql( *para\_info\_str* );

*console*.log( *result*.rows.length )

*if* (*result*.rows.length === 0 ) {

*console*.log( " exec success ");

}

*else* {

*result\_set* = "Unknown";

*console*.log( " exec faliure ");

}

})

*return* true;

}