选课与成绩管理系统

程序说明书



班级：2017211309

姓名：曹景行 钟健 梁家旭

学号：曹景行-2017211432

钟健-2016211348

梁家旭-2017211408

日期：2019-12-22

# 分工合作

### 分工内容

### 曹景行进行服务器端的开发，钟健负责客户端窗口的设计，梁家旭负责服务器端到客户端的数据传输。

### 分工率

### 曹景行33%，钟健33%，梁家旭33%。

1. 服务器端的设计
2. 设计要求
3. 数据结构与类设计
4. 实现思路
5. 实现方法
6. 客户端的设计
7. 设计要求
8. 开发环境
9. 实验思路
10. 实现方法

服务器端的设计

1. 设计要求

设计一个成绩管理系统，实现以下几个功能

* + -老师、学生的登录
  + -显示老师学生的基本信息
  + -支持学生选修必修课、查看所有必修课
  + -支持学生查看成绩
  + -支持老师查看学生成绩、修改某个学生成绩
  + -实现所有数据的文件存取功能
  + -实现排序功能
  + -增加输入异常处理
  + 基于socket API为软件增加了网络访问功能，即实现C/S模型
  + 基于select()函数，增加多用户访问和操作觉果实时同步的功能
  + 提升安全性，保障服务端数据的安全

1. 数据结构与类设计
2. 老师&学生类设计

需要明确的是我们一共有四种类型的数据需要存储：老师、学生、选修课和必修课。

课程

学生

老师

查看成绩

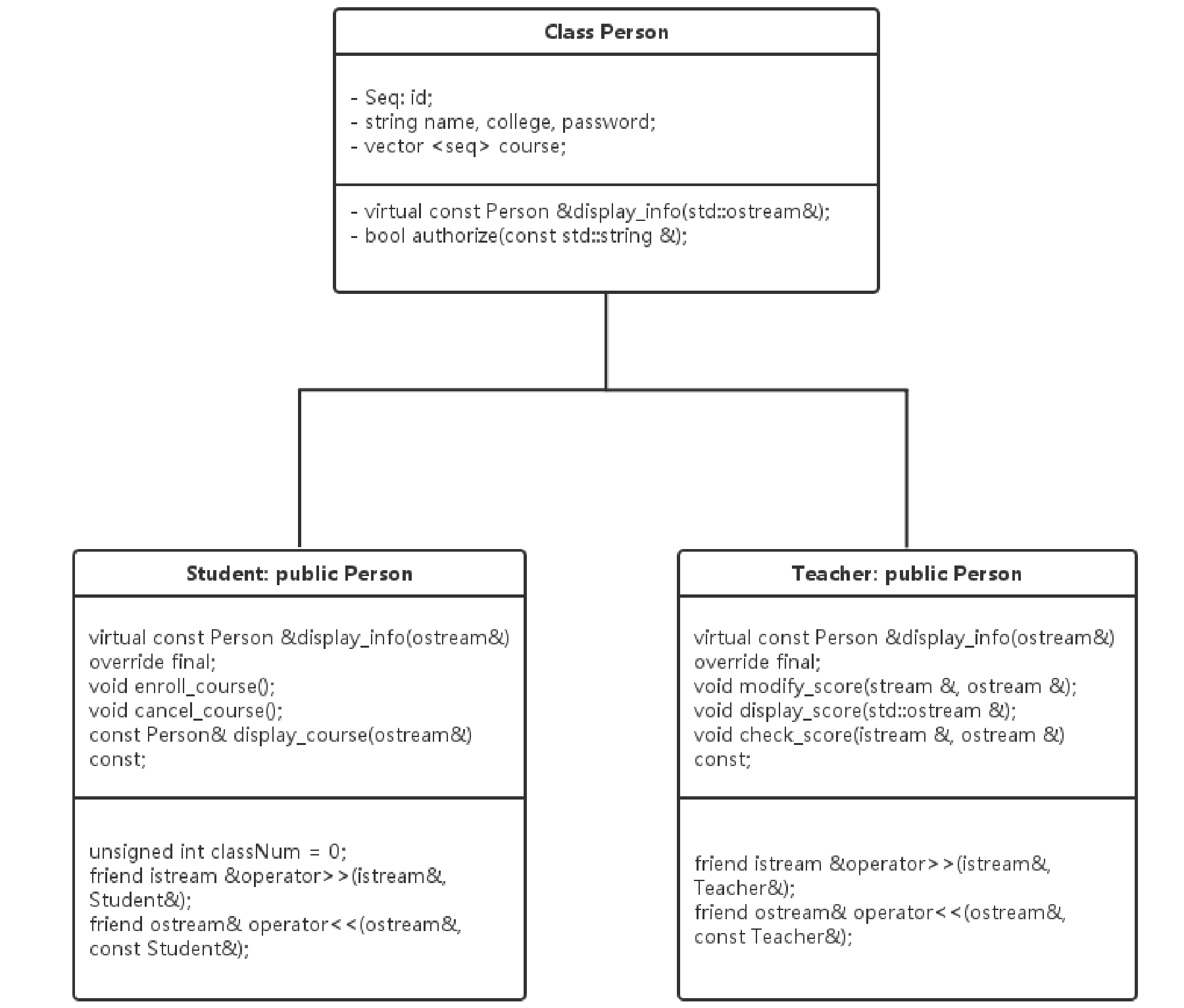
选课

ኞ਍፡ັ

查看学生成绩

修改学生成绩

基于以上特征，我们在用户类与课程类里面都存出一个与之存在对应关系的对象的下标。具体类设计如下：



1. 课程类设计

课程类中一共有两个子类：必修课与选修课类。他们都共同享有课程号，授课老师，学生成绩等成员变量，也都有显示课程基本信息，获得学生成绩等成员函数。

将二者不同之处用虚函数以及其重载来实现，最终课程类如下：

Class Course

public

:

typedef

long

seq;

typedef

unsigned

short

score;

protected

:

seq

id =

0

;

std

::

string

name;

Person

::

seq

teacher;

score

credit =

0

;

std

::

map

<

seq

,

score

>

studentScore;

public

:

virtual

double

get\_gpa(…)

= 0;

virtual

void

throw\_student(…) = 0

;

virtual

void

enroll\_student(…) = 0;

virtual

bool

display(…)

const

=

0

;

void

print\_score\_table(…)

const

;

void

change\_score(…);

Class Require\_course

public

:

virtual

double

get\_gpa(

…

)

const override

;

virtual

void

throw\_student(

…

)

override

;

virtual

void

enroll\_student(

…

)

override

;

virtual

bool

display(…)

const

override

;

Class Elective\_course

public

:

virtual

double

get\_gpa(

…

)

const override

;

virtual

void

throw\_student(

…

)

override

;

virtual

void

enroll\_student(

…

)

override

;

virtual

bool

display(…)

const

override

;

1. 系统类设计

逻辑上，系统类存储有课程数据地址与课程号、用户数据地址与用户号的映射关系，同时也是整个软件中存储系统的基础。

|  |
| --- |
| class Result\_system  { private:  Result\_system();  static std::unique\_ptr<Result\_system> m\_instance;  public:  ~Result\_system();  static Result\_system& get\_instance() //must be reference  {  if (m\_instance == nullptr)  m\_instance.reset(new Result\_system);  return \*m\_instance;  } public:  Person\_ptr get\_person(const Person::seq &); Course\_ptr get\_course(const Course::seq &);  void print\_available\_course(const Student&, std::ostream &); private: std::map <Person::seq, Person\_ptr> num\_to\_person; std::map <Course::seq, Course\_ptr> num\_to\_course; }; |

1. 实现思路
2. 文件存取
   1. 老师、学生、选修课、必修课需要存储到不同的文件下。
   2. 所有类的实例对象在程序结束时候需要调用各自析构函数释放内存。

于是考虑在基类设计一个析构函数的结构，其子类去重写对应的析构函数。然后再析构函数中，实现不同类存储到不同文件下的功能。以Student类为例，其对应的析构函数代码为：

|  |
| --- |
| Student::~Student()  {  using std::endl;  /\*store the data \*/  static std::ofstream out\_file("student.txt");  out\_file << id << ' ' << password << ' ' << name << ' ‘ << college << ' ' << classNum << endl; copy(course.begin(), course.end(), std::ostream\_iterator<int>(out\_file, " ")); out\_file << endl;  //destory vector course.clear();  } |

具体析构过程如下：在程序结束运行的时候，Result\_system的静态成员变量m\_instance被释放。由于它是一个智能指针，将自动调用所指向对象的析构函数。在系统类的析构函数中，将会分别释放所有用户、课程的对象，在释放内存的同时，数据也就被自动的保存到了文件中。

1. 成绩排序

考虑到对成绩排序的两种情况有所不同，

|  |
| --- |
| const Person& Student::display\_info(std::ostream &os, const Score\_mode  &mode)  {  /\* output the basic info \*/  Result\_system &system = Result\_system::get\_instance();  if (mode == INCREASE\_BY\_SCORE)  sort(course.begin(), course.end(), [&system, this](const seq &a,  const seq &b)  { return system.get\_course(a)->get\_score(this->id) < system.get\_course(b)->get\_score(this->id); }); else  sort(course.begin(), course.end(), [&system, this](const seq &a,  const seq &b)  { return system.get\_course(b)->get\_score(this->id) > system.get\_course(a)->get\_score(this->id); });  /\* output the score table \*/ } |

1. 网络功能

网络功能实现有以下两种方式：

1. 服务器端将数据传给用户，用户在本地可以直接调用display函数与用户交互。
2. 服务器端不将数据传给用户，而只把数据的处理结果发给用户。

第一种方法易于实现，只需要服务器将对应用户的类传给用户，用户在本地可以实现选课、修改成绩等功能，然后再将数据回传给服务器端。但是这种方法有两个严重问题在于：

1. 不易于实现多用户操作结果的实时同步。
2. 无法保证用户在本地操作的合法性。
3. 实现方法
4. 成绩的存储

course里面存储的是每个人的成绩，是考虑到老师需要修改成绩时候的方便。

1. ID的存储
   1. 主关键字使用size——t类型
   2. 使用std：：map来存储住关键字与其对应对象指针的映射。
2. Person类&Course类
   1. 使用虚函数和protected标签的成员以实现继承。使用protected的成员变量保证了其子类拥有父类成员变量的访问与修改权限。、
   2. 简化了数据结构的逻辑复杂度。
3. Result\_system类的智能指针的的单例模式实现
   1. 使用unique\_ptr只想全局唯一的静态私有成员，保证全局只有一个实例化的system类。
   2. 使用unique\_ptr有两个好处：
      1. 保证了全局唯一
      2. 在主函数结束的时候会自动调用析构函数
      3. get\_instance()返回的是引用，保证了在结束子函数的时候system不被销毁。
4. socket编程

在创建socket的子函数中，使用了《深入理解计算机系统》一书中Lab的代码，该代码提供了一些很实用的特性，例如：

|  |
| --- |
| /\*  \* open\_listenfd - open and return a listening socket on port \* Returns -1 and sets errno on Unix error.  \*/  /\* $begin open\_listenfd \*/ int open\_listenfd(int port)  {  int listenfd, optval=1; struct sockaddr\_in serveraddr;    /\* Create a socket descriptor \*/  if ((listenfd = socket(AF\_INET, SOCK\_STREAM, 0)) < 0) return -1;    /\* Eliminates "Address already in use" error from bind. \*/ if (setsockopt(listenfd, SOL\_SOCKET, SO\_REUSEADDR, (const void \*)&optval , sizeof(int)) < 0) return -1;  /\* Listenfd will be an endpoint for all requests to port on any IP address for this host \*/  bzero((char \*) &serveraddr, sizeof(serveraddr));  serveraddr.sin\_family = AF\_INET;  serveraddr.sin\_addr.s\_addr = htonl(INADDR\_ANY); serveraddr.sin\_port = htons((unsigned short)port);  if (bind(listenfd, (SA \*)&serveraddr, sizeof(serveraddr)) < 0) return -1;  /\* Make it a listening socket ready to accept connection requests \*/  if (listen(listenfd, LISTENQ) < 0)  return -1; return listenfd;  }  /\* $end open\_listenfd \*/ |

在业界socket网络编程中实现有两种主流方法，第一种是用fork()多线程来处理不同用户的请求，第二种是使用select()函数每次取可读的socket进行处理。由于多线程的处理方法效率低，在业界一般很少使用，所以采用了后者实现多用户，核心代码如下：

|  |
| --- |
| void Server::run()  {  while (1)  {  memcpy(&rfds, &fds, sizeof(fds));  if (Select(maxfd, &rfds, 0, 0, NULL) <= 0) break;    /\* if a new client come \*/ if (FD\_ISSET(listenSock, &rfds))  {  dataSock = Accept(listenSock, 0, 0);  FD\_SET(dataSock, &fds);  if (maxfd <= dataSock) maxfd = dataSock + 1;  client[dataSock].status = NO\_LOGIN; }  /\* Process the request from clients \*/ for (int fd = 0; fd < maxfd; fd++)  {  if (fd !=listenSock && FD\_ISSET(fd, &rfds))  {  if (isclosed(fd))  {  Close(fd); FD\_CLR(fd, &fds);  client[fd].status = NO\_LOGIN;  }  else receive\_data(fd);  }  }  }  Close(listenSock);  Close(dataSock);  } |

服务器端日志文件

通过使用recvfrom()函数可以获得客户端的IP地址与端口号，基于这个函数我在服务器端实现了一个简单的日志记录功能，能够同时输出在命令行合同目录下的./server.log中。

日志函数的实现可以方便发现不良的访问请求，也可以记录不同用户的活动。

健全的权限管理

考虑到本系统是数据管理系统，安全性是关键。本程序实现了不同层次的安全性处理：

1. 用户需要使用用户名和密码来登录。每一个账户都有“教师”与“学生”的标示，象征着用户的不同权限。
2. 客户端限制允许操作。具体而言就是学生之能够查看选修课、选课、退课等操作，而教师只有查看学生成绩、修改成绩的权限。对有越权的操作将会报错。
3. 服务器检查用的身份和请求类型。服务器端收到一个请求的时候，首先检查该用户是否有使用该请求的对应权限。如果没有则否决这次请求并返回错误信息。

客户端的设计

1. 实验要求

设计一个图形化界面的成绩管理系统，实现以下几个功能：

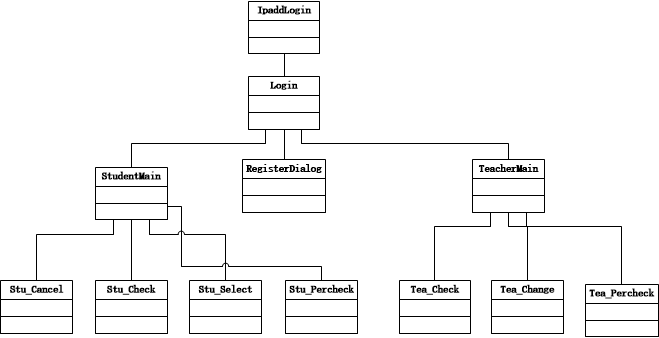
* -老师、学生的登录
* -显示老师学生的基本信息
* -支持学生选修必修课、查看所有必修课
* -支持学生查看成绩
* -支持老师查看学生成绩、修改某个学生成绩

1. 开发环境

使用qt5.13版本进行开发

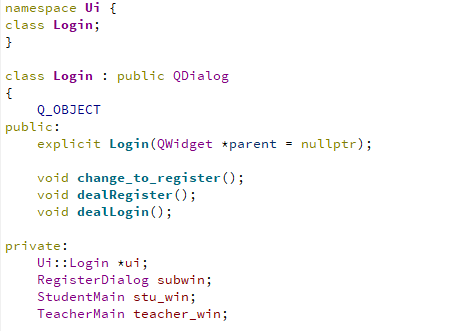
1. 实验思路

服务端整体的代码类的结构如下，其中根节点与子结点的关系是——根节点的类的成员变量中有叶节点的类，是包含关系。

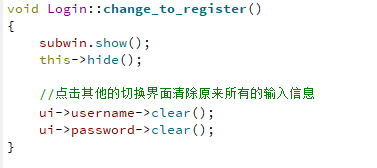


1. 实验方法
2. 页面切换

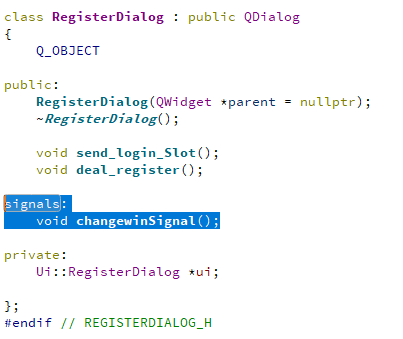
以Login类为例，Login的成员变量中分别有StudentMain,RegisterDialog,TeacherMain的子窗口



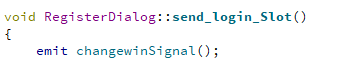
Login类在想跳转到成员变量类的窗口时，只需在连接点击按钮的函数中使用this.hide(),子窗口.show()即可，如下



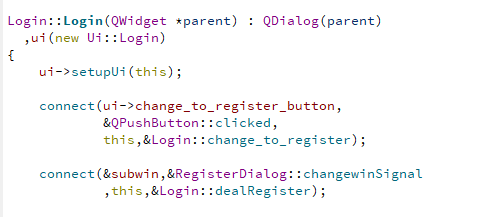
而子类窗口要实现跳转到Login类窗口，需要在子类中声明一个信号

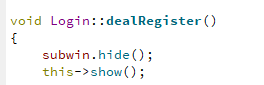


子类在接收鼠标点击按钮后发送信号



主类Login在接收到信号后，进行子窗口.hide(),this.show()的操作

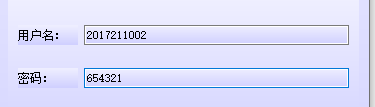




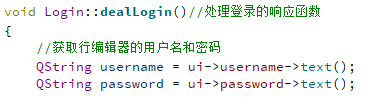
如此实现了主子页面的切换

1. 获取用户的输入

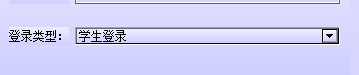
服务器端页面输入有两种，第一是使用Line Edit接收输入，如下



此时使用LineEdit的自带的text()函数可获取到用户的输入



第二种输入是下拉框，如下

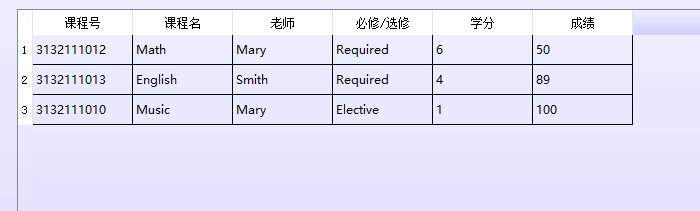


此时使用判断下拉框当前页号可以判断出用户选择了那一个类型



C:数据的显示

本客户端数据的显示常采用表格形式战术，如下



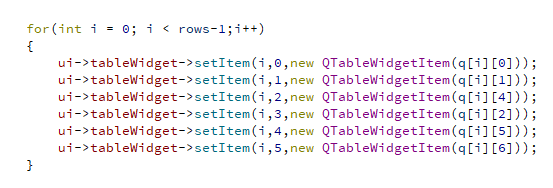
表格的显示是先设置表格的最上部属性名字，这个直接在QT设计界面设计，如下



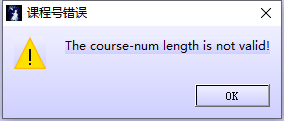
然后设计表格的总行号



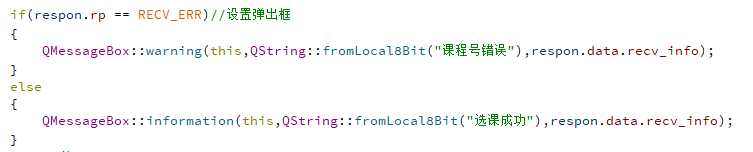
最后逐项获取每一行每一列的值，并写入表格中即可



D．弹出框

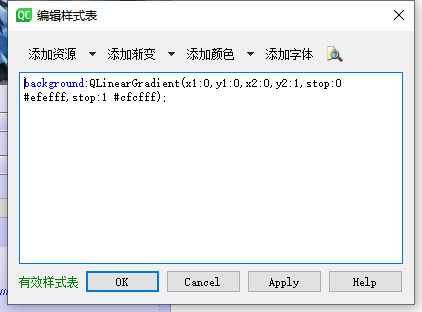


此种类型的弹出框在鼠标点击按钮后判断输入的格式、类型是否正确，若不正确弹出警告框并打印错误信息，若正确弹出通知框，实现代码如下



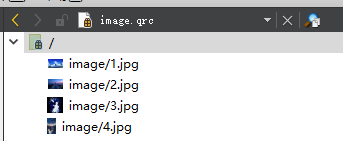
E.界面风格

所有的主界面风格均采用如下的渐变式风格

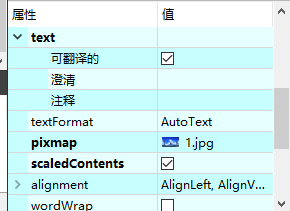


F.附带背景图片

首先添加resource文件，然后添加图片



然后创建一个QLabel对象，在QLabel的pixmap中选择图片即可



分工合作

1. 分工内容

曹景行进行服务器端的开发，钟健负责客户端窗口的设计，梁家旭负责服务器端到客户端的数据传输。

1. 分工率

曹景行33%，钟健33%，梁家旭33%。