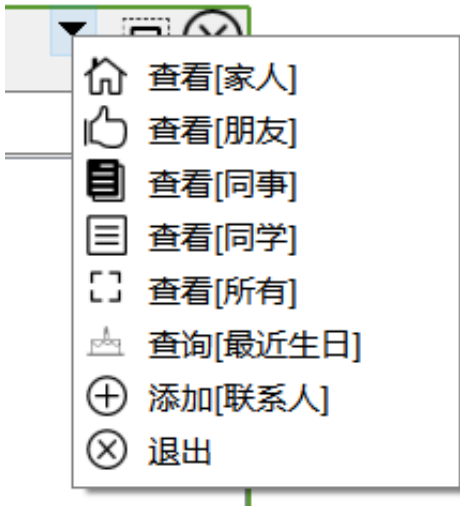


添加菜单



1. 添加 **QToolButton** 控件，清除**text**，**icon** 为空，设置属性 **arrowType** 为 ****DownArrow****，这样就变成一个倒三角形状
2. 弹出菜单
 1. 添加菜单
 2. 设置样式（默认的有点难看，虽然下面的代码设置之后也好看不到哪里去）
 3. 关联菜单信号和槽函数

```
1. QMenu* popMenu = new QMenu(this);
2. popMenu->addAction(new QAction(QIcon(":/new/icon/icon_home.png"), "
   查看[家人]", this));
3. popMenu->setStyleSheet("QMenu{background:white;border:1px solid gra
   y;padding:5px}"
4.                               "QMenu::item{padding:0px 40px 0px 30px;he
   ight:25px}"
5.                               "QMenu::item:selected{background:lightblu
   e;color:white;}"
6.                               "QMenu::separator{height:1px ;background:
   lightgray;margin:5px,0px,5px,0px;}");
7. connect(popMenu,&QMenu::triggered,this,&MainWindow::do_menu_trigger
   ed);
8. popMenu->exec(QCursor::pos()); // 菜单出现的位置为当前鼠标的位置
```

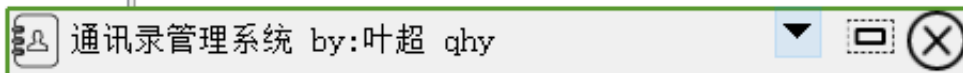
3. 响应菜单选中信号（写槽函数）

```
1. void MainWindow::do_menu_triggered(QAction *action){
2.     if(action->text() == "查看[家人]"){ //这里直接通过菜单tex
   t来判断
3.         ui->tabWidget->setCurrentIndex(0); //选中查看[家人]
4.     }else if(action->text() == "退出"){
5.         this->close();
6.     }
7. }
```

4. 补充：弹出右键菜单

1. 从指定控件的类派生出一个类，把这个控件提升为这个派生类
2. 重写 **contextMenuEvent**，在这里创建菜单和关联信号和槽函数
3. 写对应槽函数

重写界面边框



原理实际就是去掉边框，然后添加label 和 按钮控件手动模拟

1. 去掉系统默认边框的显示

```
1. this->setWindowFlags(Qt::Window|Qt::FramelessWindowHint |Qt::WindowSystemMenuHint|Qt::WindowMinimizeButtonHint|Qt::WindowMaximizeButtonHint);
```

2. 界面布局

1. 左边添加一个 **label** 控件，去掉text，设置 **pixmap** 属性，来设置图片
2. 中间放一个label空间，然后是3个按钮控件
3. 设置对应的图片，这里用QSS样式设置
4. **QSS样式设置**

```

1. //最小化按钮
2. QPushButton {border-image: url(:/new/icon/mini.png);}
3. QPushButton:hover{
4.     background-color: qconicalgradient(cx:0.5, cy:0.522909, angle:179.9
       , stop:0.494318 rgba(134, 198, 233, 255), stop:0.5 rgba(206, 234, 2
       48, 255));
5.     border-radius:5px;
6. }
7. QPushButton:pressed{
8.     background-color:#CCCCCC;
9.     border-radius:5px;
10.    border: 1px solid #5F92B2;
11. }
12. //关闭按钮
13. QPushButton {
14.     border-image: url(:/new/icon/close.png);
15. }
16. QPushButton:hover {
17.     background-color: #FF0000;
18.     border-radius:5px;
19.     border: 1px solid #5F92B2;
20. }
21. QPushButton:pressed{
22.     background-color:#CCCCCC;
23.     border-radius:5px;
24.     border: 1px solid #5F92B2;
25. }

```

3. 代码实现

1. 响应3个按钮点击信号，弹出菜单前面已经提到

```

1. this->showMinimized(); //最小化窗口
2. this->close();          //关闭窗口

```

2. 按住标题移动功能，重写 **mousePressEvent** **mouseReleaseEvent** **mouseMoveEvent** 事件

```

1. void MainWindow::mousePressEvent(QMouseEvent * e)
2. {
3.     last = e->globalPos();
4. }
5. void MainWindow::mouseReleaseEvent(QMouseEvent * e)
6. {
7.     int dx = e->globalX() - last.x();
8.     int dy = e->globalY() - last.y();
9.     move(x()+dx, y()+dy);
10. }
11. void MainWindow::mouseMoveEvent(QMouseEvent * e)
12. {
13.     int dx = e->globalX() - last.x();
14.     int dy = e->globalY() - last.y();
15.     last = e->globalPos();
16.     move(x()+dx, y()+dy);
17. }

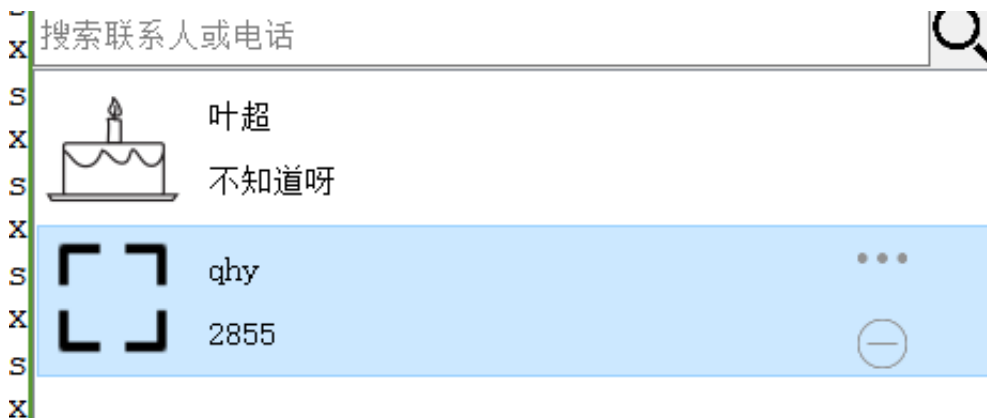
```

搜索功能，编辑框虚字提示



1. 设置编辑框属性 **placeholderText**，即可实现上面的虚字提示功能，（有输入虚字提示消失）
2. 搜索按钮 和 上面最小化 按钮 QSS设置一样

ListWidget 控件自定义窗口



思路：**ListWidget** 控件下的每一个 Item (**QListWidgetItem**) 都可以看作一个窗口，只要把窗口设置成自定义的窗口即可

1. 创建子窗口，关联子窗口和ListWidget，关联子窗口和父窗口的信号和槽函数

```

1.      List_Item_Form *custom1 = new List_Item_Form(addressbook.Item[i]);//
      创建窗口
2.      QListWidgetItem *listItem1 = new QListWidgetItem();           //
      创建一个listItem
3.      listItem1->setSizeHint(QSize(0, 71));                           //
      设置大小
4.      ui->listWidget->addItem(listItem1);                             //
      把创建的ListItem加入ListWidget
5.      ui->listWidget->setItemWidget(listItem1, custom1);               //
      设置窗口
6.      connect(custom1,List_Item_Form::list_item_form_close,this,do_list_it
      em_form_close);//关联子窗口和父窗口的信号和槽函数，这里是子窗口删除信号

```

2. 槽函数之删除子窗口

1. 判断信号子窗口
2. 使用 **takeItem** 来去除子窗口

```

1.      void MainWindow::do_list_item_form_close(Person *p)
2.      {
3.          //这里通过值一样的 p属性来判断是哪个窗口发出信号
4.          int n = ui->listWidget->count();
5.          for(int i = 0 ; i < n ; ++i){
6.              List_Item_Form *pp =(List_Item_Form *) ui->listWidget->itemWidg
              et( ui->listWidget->item(i));
7.              if( pp->p == p ){
8.                  ui->listWidget->takeItem(i);
9.                  break;
10.             }
11.         }
12.     }

```

3. 一些函数

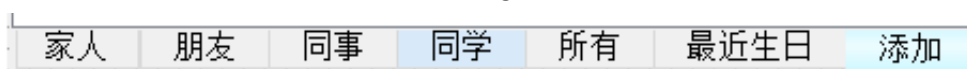
```

1.      //隐藏和显示 QListItem
2.      ui->listWidget->item(i)->setHidden(true/false);
3.      //获取指定行的QlistItem,(item函数)
4.      ui->listWidget->item(i)
5.      //由制定QlistItem获取窗口(itemWidget函数)
6.      List_Item_Form *pp =(List_Item_Form *) ui->listWidget->itemWidget();

```

TabWidget 控件

这里实际当按钮使用，使用 *TabWidget* 方便创建多个按钮和响应时间



1. QSS样式

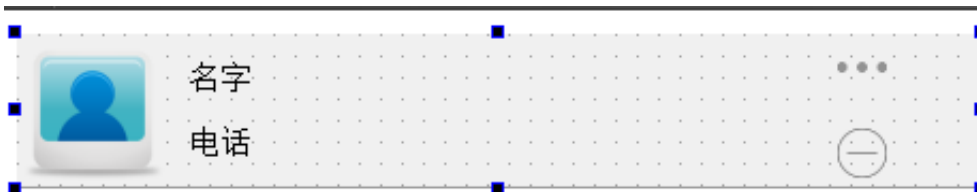
```

1. QTabBar::tab:selected {
2.     background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
3.     stop: 0 #caf7ff, stop: 0.4 #ebfcff,
4.     stop: 0.5 #ebfcff, stop: 1.0 #caf7ff);
5. }

```

2. 添加槽函数 `void MainWindow::on_tabWidget_currentChanged(int index)`

ListItem子窗口



1. 设置label 的图片 和 按钮的样式

```

1. QToolButton {
2.     border-image: url(../new/icon/more.png);
3.     border-radius: 2px;
4.     border: 1px solid rgb(89, 153, 48);
5.     background: transparent;
6.     color: green;
7. }
8. QToolButton:hover{
9.     background-color: #86C6E9;
10.    border-radius: 5px;
11.    border: 1px solid #5F92B2;
12. }
13. QToolButton:pressed{
14.     background-color: #CCCCCC;
15.     border-radius: 5px;
16.     border: 1px solid #5F92B2;
17. }

```

2. 移入显示按钮，移出隐藏按钮（重写 `enterEvent` `leaveEvent`）

```

1. void List_Item_Form::enterEvent(QEvent *)
2. {
3.     ui->toolButton->setVisible(true);
4.     ui->toolButton_2->setVisible(true);
5. }
6. void List_Item_Form::leaveEvent(QEvent *)
7. {
8.     ui->toolButton->setVisible(false);
9.     ui->toolButton_2->setVisible(false);
10. }

```

3. 弹出（创建）窗口

```
1. Info_Form *k = new Info_Form(p); //创建窗口
2. connect(k, Info_Form::delete_Item, this, do_delete_Item); //关联信号
3. k->show(); //显示窗口
```

4. 日期的计算和选择

- 后来了解到有一个 fromstring 的函数什么的，应该不用下面那么麻烦...

```
1. //日期计算和转换
2. flg_recent_birth = 0;
3. //判断是否最近生日
4. QString str = p->GetBirth().c_str(); //获取日期 1996-2-1 形式
5. QStringList list1 = str.split('-'); //分离 字符串
6. QDate birth_1 = QDate::currentDate(); //当前日期
7. QDate birth_2 = birth_1.addDays(7); //7天之后的日期
8. //把时间设置为今年，用以判断是否最近生日
9. QDate birth(birth_1.year(), QString(list1.at(1)).toInt(), QString(list1.at(2)).toInt()); //转换今年的生日日期
10. if(birth >= birth_1 && birth <= birth_2){ //判断是否在最近7天
11.     flg_recent_birth = 1;
12.     ui->label_4->setText(birth.toString("yyyy-M-d"));
13. }
```

窗口编辑和查看切换

编辑模式

 查看详细信息或添加联系人



名字：

电话号码：

邮箱：

编辑资料

邮政编码：

保存

地址：

取消

生日：

认识地点：


家人

朋友

同事

同学

查看模式

 查看详细信息或添加联系人



名字：

电话号码：

邮箱：

编辑资料

邮政编码：

保存

地址：

关闭

生日：

2016-6-4

认识地点：

家人

朋友

同事

同学

思路：

1. 通过 **setEnabled** 函数来设置是否可编辑
2. 通过 **setStyleSheet** 函数来改变显示的样式


```

1. ui->dateEdit->setEnabled(true);
2. ui->lineEdit->setStyleSheet("QLineEdit {border: 1px solid rgb(0, 0, 0);background:White;color: Black;}");
3. ui->lineEdit->setStyleSheet("QLineEdit {border: 0px;background:transparent;color: Black;}");

```

3. 禁用控件之后，字体会变灰（包括上面的Tab控件也是这样来避免禁用后难看的问题）

- 通过 样式表里面的 color 属性来设置字体，避免字体变灰

4. 编辑框边框隐藏

- 通过设置样式表里面的 border 属性大小为 0px 来隐藏.边框

5. TabWidget 设置Tab按钮的样式

```

1. QTabWidget,QWidget{
2. border:0px;
3. background-color: rgb(240, 240, 240);
4. }
5. QLineEdit,QLabel,QTabBar{
6. color:#000000
7. }
8. QTabBar::tab:selected {
9. background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
10. stop: 0 #caf7ff, stop: 0.4 #ebfcff,
11. stop: 0.5 #ebfcff, stop: 1.0 #caf7ff);
12. }

```

6. DataWidget 控件，设置 上面的样式并没有 去除边框 和 黑色背景（它是不是有子控件？）

- 所以使用一个label控件来显示查看模式，编辑模式才显示DataEdit控件

```

1. //设置日期弹出日期框
2. ui->dateEdit->setCalendarPopup(true);
3. ui->lineEdit_6->setVisible(false);
4. ui->dateEdit->setVisible(true);

```

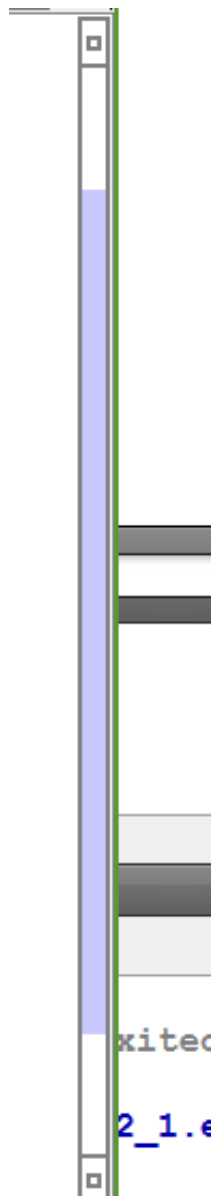
弹出文件选择框

```

1. QFileDialog *fileDialog = new QFileDialog(this);
2. fileDialog->setWindowTitle(tr("Save As"));
3. fileDialog->setAcceptMode(QFileDialog::AcceptOpen);
4. fileDialog->setFileMode(QFileDialog::AnyFile);
5. fileDialog->setViewMode(QFileDialog::Detail);
6. fileDialog->setGeometry(10,30,300,200);
7. fileDialog->setDirectory(".");
8. if(fileDialog->exec() == QDialog::Accepted) {
9.     avastar_path = fileDialog->selectedFiles()[0];
10.     QMessageBox::information(NULL,"路径",path);
11. }

```

ListWidget 样式



```
1.  QScrollBar:vertical {
2.      border: 2px solid grey;
3.      background: white;
4.      width: 15px;
5.      margin: 22px 0 22px 0;
6.  }
7.  QScrollBar::handle:vertical {
8.      background: rgb(229,243,255);
9.      min-height: 20px;
10. }
11. QScrollBar::handle:hover:vertical {
12.     background: rgb(205,232,255);
13.     min-height: 20px;
14. }
15. QScrollBar::handle:vertical:pressed {
16.     background: rgb(200,200,255);
17.     min-height: 20px;
18. }
19. QScrollBar::add-line:vertical {
20.     border: 2px solid grey;
21.     background: white;
22.     height: 20px;
23.     subcontrol-position: bottom;
24.     subcontrol-origin: margin;
25. }
26. QScrollBar::add-line:vertical:hover {
27.     border: 2px solid grey;
28.     background: rgb(231,231,231);
29.     height: 20px;
30.     subcontrol-position: bottom;
31.     subcontrol-origin: margin;
32. }
33. QScrollBar::add-line:vertical:pressed {
34.     border: 2px solid grey;
35.     background: rgb(206,206,206);
36.     height: 20px;
37.     subcontrol-position: bottom;
38.     subcontrol-origin: margin;
39. }
40. QScrollBar::sub-line:vertical {
41.     border: 2px solid grey;
42.     background: white;
43.     height: 20px;
44.     subcontrol-position: top;
45.     subcontrol-origin: margin;
46. }
47. QScrollBar::sub-line:vertical:hover {
48.     border: 2px solid grey;
49.     background: rgb(231,231,231);
50.     height: 20px;
51.     subcontrol-position: top;
52.     subcontrol-origin: margin;
53. }
```

```

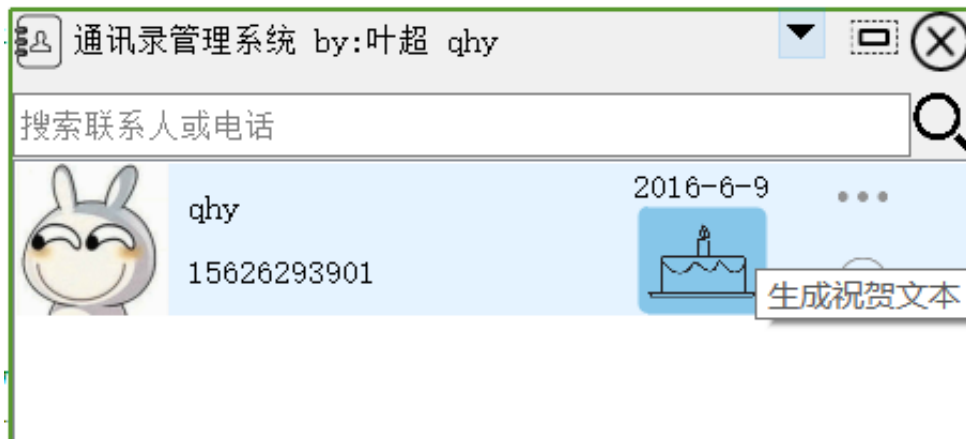
54.     QScrollBar::sub-line:vertical:pressed {
55.         border: 2px solid grey;
56.         background: rgb(206,206,206);
57.         height: 20px;
58.         subcontrol-position: top;
59.         subcontrol-origin: margin;
60.     }
61.     QScrollBar::up-arrow:vertical, QScrollBar::down-arrow:vertical {
62.         border: 2px solid grey;
63.         width: 3px;
64.         height: 3px;
65.         background: white;
66.     }
67.     QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical {
68.         background: none;
69.     }

```

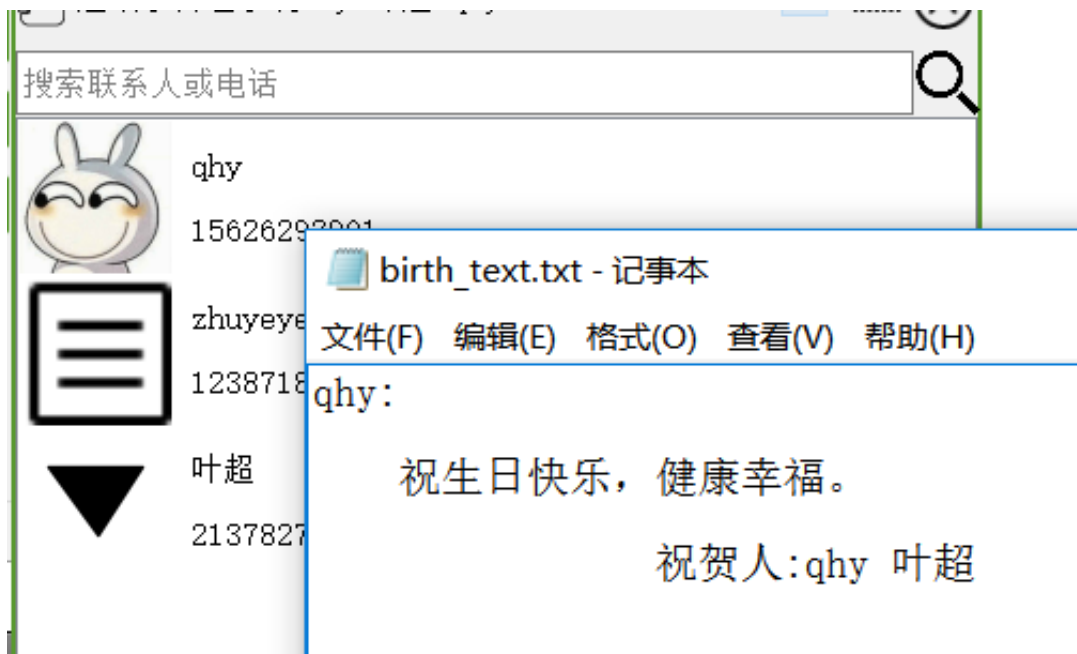
生日提醒

界面提示

只需判断一下是否在最近生日，来控制是否显示那个控件即可，前面已经提到日期的计算



生成祝贺文本



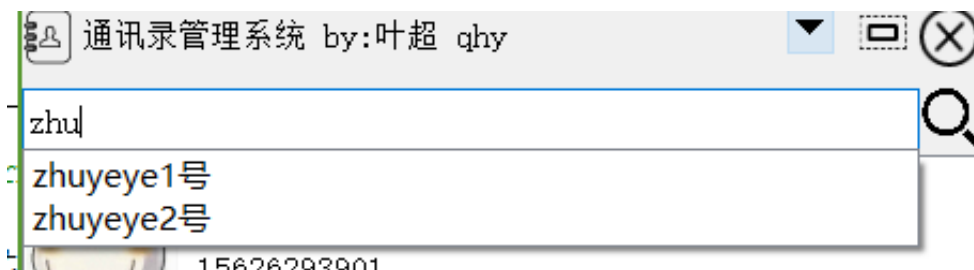
代码

```

1. //生成祝贺文本文件
2. string birth_file_name("birth_text.txt");
3. fstream file(birth_file_name,ios::out);
4. string birth_text = p->GetName() + ":\n\n    祝生日快乐，健康幸福。\n\n\
   t\t祝贺人:qhy 叶超\n";
5. file<<birth_text;
6. file.close();
7. //使用notepad打开文本
8. QString program = "notepad";
9. QStringList arguments;
10. arguments << birth_file_name.c_str();
11. QProcess *myProcess = new QProcess(this);
12. myProcess->start(program, arguments);
13. //控制台下，可以直接用 system("notepad 文件名")来运行，界面下会弹出黑框
    框，所以不用

```

搜索编辑框自动补全功能



使用 QCompleter 来完成

注意：增加联系人之后要更新提示的列表

```

1. //自动补全单词列表
2. QStringList cplist;//单词列表
3. QCompleter * cp;//补全器
4. QStringListModel *string_list_model;//单词列表模型，用以更新补全器的列表
5.
6. //Edit控件关联Qcompleter对象
7. cplist.append(addressbook.Item[i]->GetName().c_str());//把每组数据的名字
   和电话加入List中，用于搜索编辑框的自动补全功能
8. cplist.sort();//先对List里面的字符串进行排序
9. //添加自动补全 completer 到 QLineEdit
10. QCompleter * cp = new QCompleter(string_list_model,this);
11. ui->lineEdit->setCompleter(cp);
12. string_list_model->setStringList(cplist);
13. //增加单词，前面已经关联了string_list_model,这里就没补全器什么事了..
14.     cplist.append(str);
15.     cplist.sort();
16.     string_list_model->setStringList(cplist);

```

添加双击显示详细信息的功能


- 重写 QListWidgetItem 的 **mouseDoubleClickEvent** 事件即可
- 不是在主窗口的 **ListWidget** 响应双击事件

```

1. void List_Item_Form::mouseDoubleClickEvent(QMouseEvent *)
2. {
3.     on_toolButton_clicked();//调用显示详细信息的按钮事件即可，不需要再次复制
   一次代码
4. }

```

添加exe文件的图标icon

 cpp_project_2_1.exe

1. 把图标文件icon.ico复制到项目目录下
2. 在 .pro 文件以下一下语句

```
1. RC_ICONS = icon.ico
```