

知识点1【绘图的概述】

案例：画一个背景图

案例：通过update()重新加载绘图事件

知识点2【画家的其他绘制函数】

1、划线drawLine

2、画矩形

3、画圆

知识点3【绘图设备】

案例1：QBitmap

案例2：image的像素操作

案例：重现绘图指令QPicture

知识点4【项目篇】

项目1：工业控制的UI（必须完成）

1、注册登录界面：

2、主页面 柱状图 创新

3-1设置页面

3-2：操作历史记录

3-3：密码设置

3-4：串口调试

项目2：棋牌游戏(尽量完成+加分项)

1、规则介绍：

2、功能需求：

1) 基本功能:

1) 拓展创新:

项目一：补充知识点

1、窗口内切换 多个窗口 (QStackWidget)

知识点1 【绘图的概述】

图系统基于QPainter, QPainterDevice和QPaintEngine三个类

QPainter (画家) 使用QPaintEngine (绘图工具) 在 QPainterDevice (绘图设备) 上画画。



注意:

- 1、如果在主窗口上绘画 必须在绘图事件 (paintEvent) 中完成画画.
- 2、绘图事件 调用的时机 (1、窗口加载 2、update())

内容索引书签搜索

索引

查找(L):

QWid

accessibility for QWidget Applica...

QWidget

QWidget::DrawChildren

QWidget::DrawWindowBackground

QWidget::IgnoreMask

QWidget::RenderFlag

QWidget::RenderFlags

QWidgetAction

QWidgetItem

QWIDGETSIZE_MAX

~QWidget

~QWidgetAction

~QWidgetItem

Reimplemented Protected Functions

```
virtual bool event(QEvent *event)
virtual void initPainter(QPainter *painter) const
virtual int metric(PaintDeviceMetric m) const
```

- 9 protected functions inherited from `QObject`
- 1 protected function inherited from `QPaintDevice`

Protected Slots

```
void updateMicroFocus()
```

Mouse

[virtual protected] bool QWidget::event(QEvent *event)

Reimplemented from `QObject::event()`.

This is the main event handler; it handles event `event`. You can reimplement this function in a subclass, but we recommend using one of the specialized event handlers instead.

Key press and release events are treated differently from other events. `event()` checks for Tab and Shift+Tab and tries to move the focus appropriately. If there is no widget to move the focus to (or the key press is not Tab or Shift+Tab), `event()` calls `keyPressEvent()`.

Mouse and tablet event handling is also slightly special: only when the widget is `enabled`, `event()` will call the specialized handlers such as `mousePressEvent()`; otherwise it will discard the event.

This function returns `true` if the event was recognized, otherwise it returns `false`. If the recognized event was accepted (see `QEvent::accepted`), any further processing such as event propagation to the parent widget stops.

See also `closeEvent()`, `focusInEvent()`, `focusOutEvent()`, `enterEvent()`, `keyPressEvent()`, `keyReleaseEvent()`, `leaveEvent()`, `mouseDoubleClickEvent()`, `mouseMoveEvent()`, `mousePressEvent()`, `mouseReleaseEvent()`, `moveEvent()`, `paintEvent()`, `resizeEvent()`, `QObject::event()`, and `QObject::timerEvent()`.

```
[virtual protected] void QWidget::paintEvent(QPaintEvent *event)
```

This event handler can be reimplemented in a subclass to receive paint events passed in *event*.

A paint event is a request to repaint all or part of a widget. It can happen for one of the following reasons:

- `repaint()` or `update()` was invoked,
- the widget was obscured and has now been uncovered, or
- many other reasons.

Many widgets can simply repaint their entire surface when asked to, but some slow widgets need to optimize by painting only the requested region: `QPaintEvent::region()`. This speed optimization does not change the result, as painting is clipped to that region during event processing. `QListView` and `QTableView` do this, for example.

Qt also tries to speed up painting by merging multiple paint events into one. When `update()` is called several times or the window system sends several paint events, Qt merges these events into one event with a larger region (see `QRegion::united()`). The `repaint()` function does not permit this optimization, so we suggest using `update()` whenever possible.

When the paint event occurs, the update region has normally been erased, so you are painting on the widget's background.

The background can be set using `setBackgroundRole()` and `setPalette()`

widget.h的类中:

```
class Widget : public QWidget
{
    Q_OBJECT

public:
    explicit Widget(QWidget *parent = 0);
    ~Widget();
    //重写绘图事件（声明）
    virtual void paintEvent(QPaintEvent *event);

private:
    Ui::Widget *ui;
};

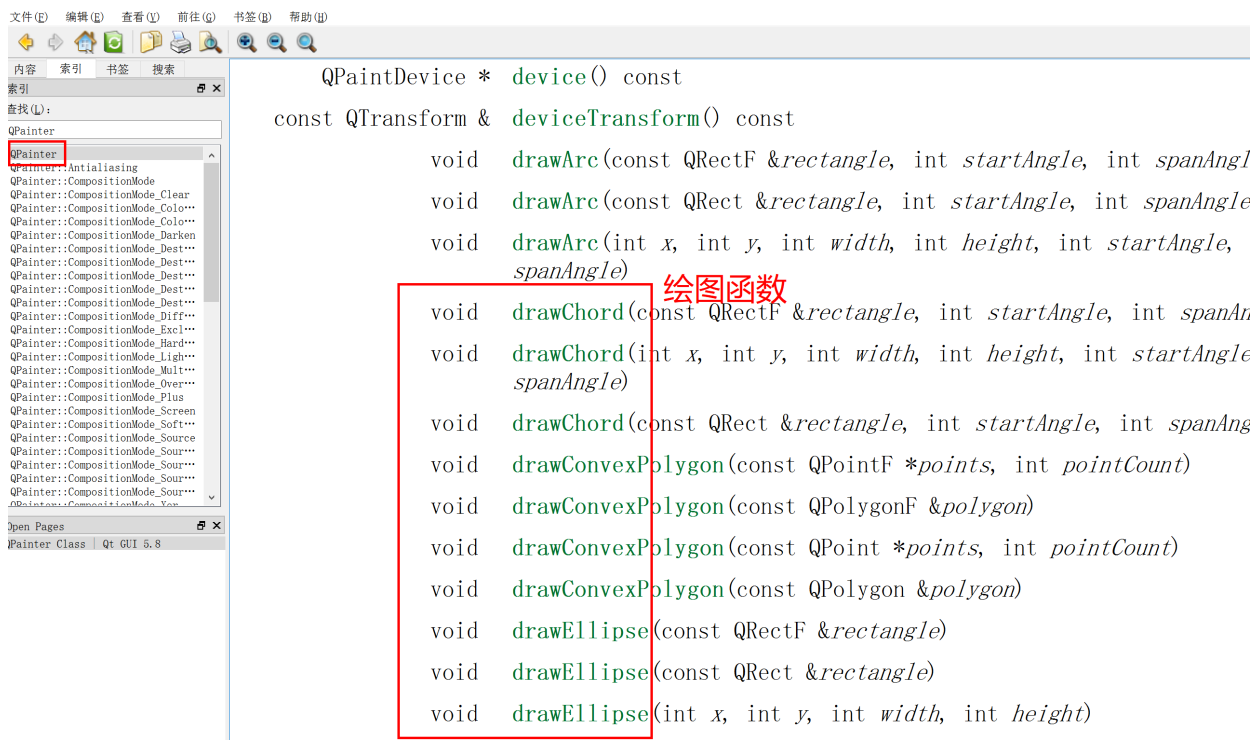
#endif // WIDGET_H
```

widget.cpp

```
15
16 //重写绘图事件|
17 void Widget::paintEvent(QPaintEvent *event)
18 {
19
20 }
21
```

案例：画一个背景图

QPainter画家的方法：



quality = -1) const
QPixmap scaled(const QSize &size, Qt::AspectRatioMode aspectRatioMode = Qt::IgnoreAspectRatio, Qt::TransformationMode transformMode = Qt::FastTransformation) const
QPixmap scaled(int width, int height, Qt::AspectRatioMode aspectRatioMode = Qt::IgnoreAspectRatio, Qt::TransformationMode transformMode = Qt::FastTransformation) const

在widget.cpp的绘图事件中：

```
1 // 重写绘图事件 主窗口加载的时候 调用
2 void Widget::paintEvent(QPaintEvent *event)
3 {
4     // 定义一个画家 画图片
5     QPainter *painter = new QPainter(this);
6
7     // 定义一个图片控件
8     QPixmap pix;
9     pix.load(":/image/Sunny.jpg");
10    // 修改图片大小(和窗口一样大)
11    pix.scaled(this->width(), this->height());
12
13    // 画家在主窗口绘画
```

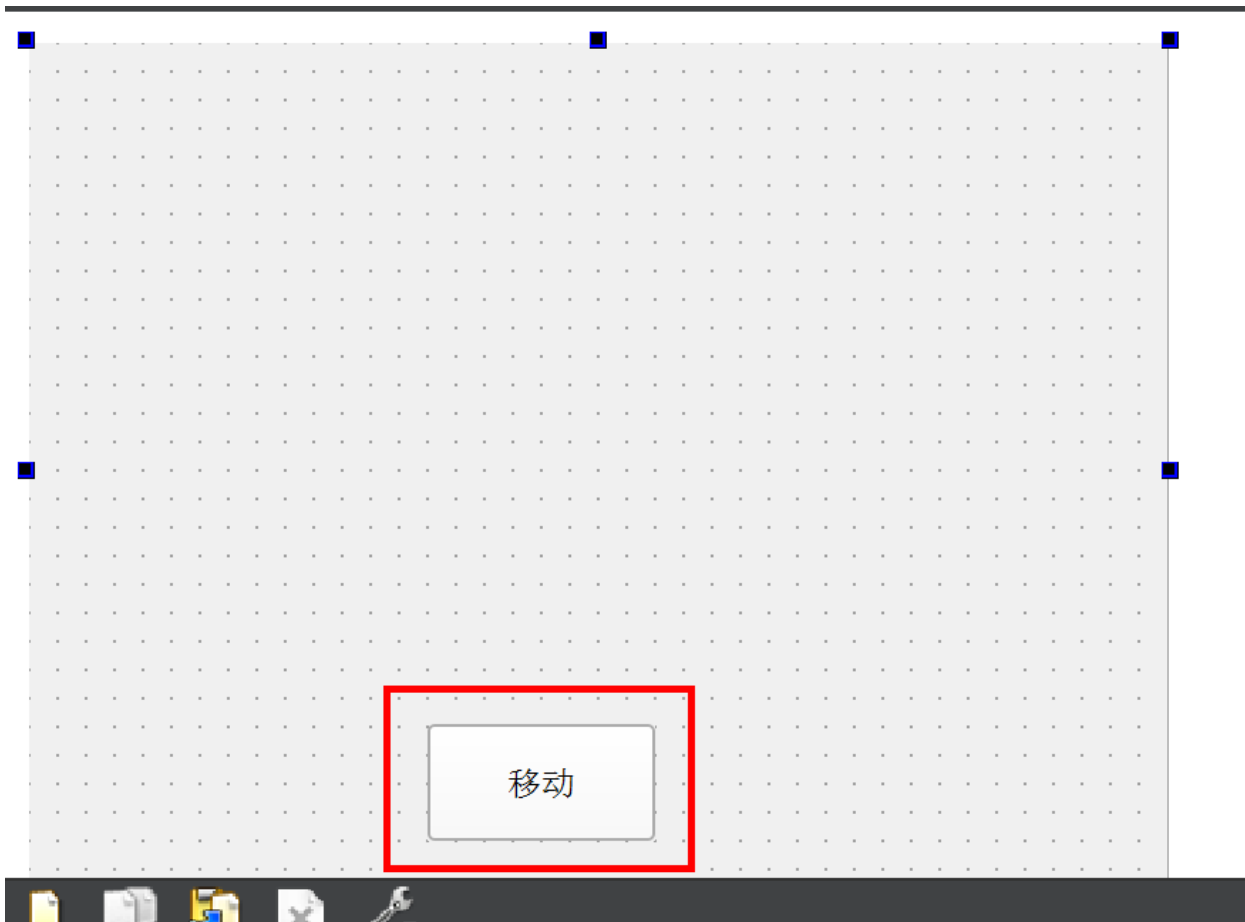
```
14 painter->drawPixmap(0,0,this->width(),this->height(), pix);  
15 }
```

运行结果：



案例：通过update()重新加载绘图事件

ui文件：



widget.h文件声明 画图时间函数

```
5
6 namespace Ui {
7     class Widget;
8 }
9
10 class Widget : public QWidget
11 {
12     Q_OBJECT
13
14 public:
15     explicit Widget(QWidget *parent = 0);
16     ~Widget();
17     //重写绘图事件
18     virtual void paintEvent(QPaintEvent *event);
19
20 private:
21     Ui::Widget *ui;
22 };
23
24 #endif // WIDGET_H
25
```

widget.cpp的构造函数中 让按钮动起来

```
Widget::Widget(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Widget)
{
    ui->setupUi(this);
    this->resize(800,600);

    //将button一定到最下方的中间位置
    //窗口的宽 this->width(); 高 this->height();
    //按钮的宽 ui->pushButton->width(); 高ui->pushButton->height()
    ui->pushButton->resize(120,80);
    ui->pushButton->move( 0.5*(this->width()- ui->pushButton->width()),\
                        this->height()-ui->pushButton->height());
    connect(ui->pushButton, &QPushButton::clicked,[=]() {

        //重新绘制加载绘图事件
        this->update();
    });
}
```

widget.cpp的paintEvent绘图事件:

```
1 //重写绘图事件
2 void Widget::paintEvent(QPaintEvent *event)
3 {
4     static int x = 0;
5     //定义一个画家
6     QPainter *painter = new QPainter(this);
7     //定义一个图片控件
8     QPixmap pix;
9     pix.load(":/image/sunny.png");
10    pix.scaled( pix.width()*0.5, pix.height()*0.5);
11
12    //画家绘图
13    //drawPixmap前两个参数是起点坐标
14
15    painter->drawPixmap(x,0, pix.width(), pix.height(),pix);
16    x+=2;
17    if(x >= this->width())
18    {
19        x = 0;
20    }
21 }
```

知识点2 【画家的其他绘制函数】

1、划线drawLine


```
void drawLine(const QLineF &line)
void drawLine(const QLine &line)
void drawLine(int x1, int y1, int x2, int y2)
void drawLine(const QPoint &p1, const QPoint &p2)
void drawLine(const QPointF &p1, const QPointF &p2)
void drawLines(const QLineF *lines, int lineCount)
```

```
1 //划线
2 painter->drawLine(0,0,200,200);
```

2、画矩形

```
void drawRect(const QRectF &rectangle)
void drawRect(int x, int y, int width, int height)
void drawRect(const QRect &rectangle)
```

//画家对 画笔进行设置

```
void setPen(const QPen &pen)
void setPen(const QColor &color) 设置画笔
void setPen(Qt::PenStyle style)
```

设置画笔颜色



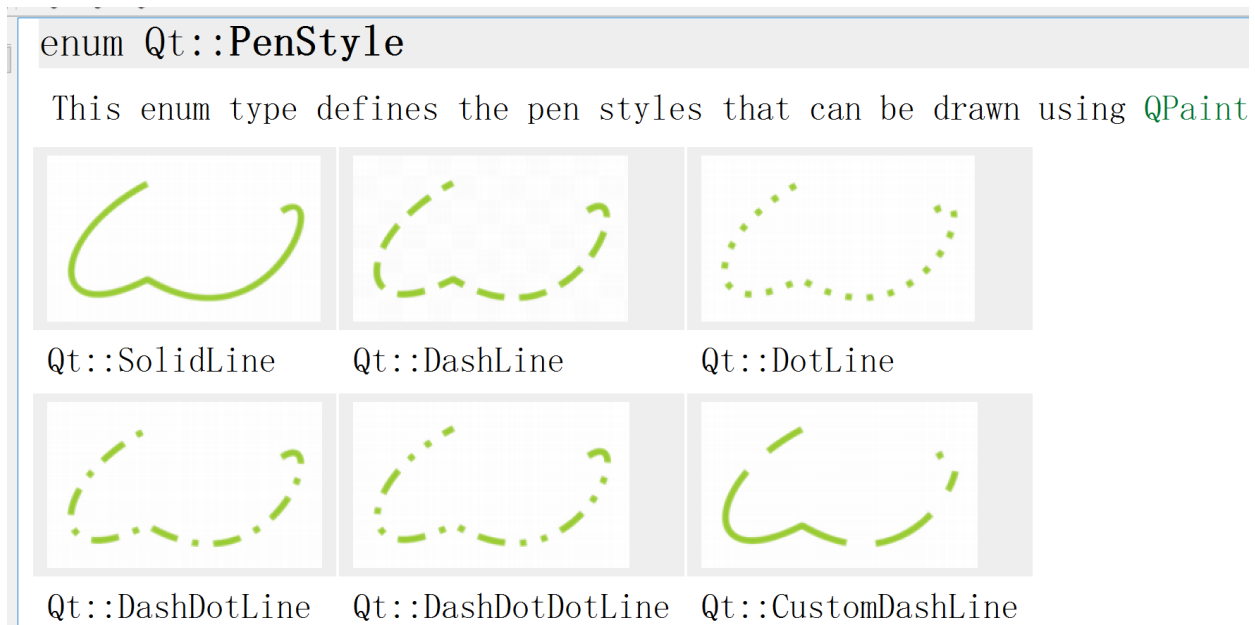
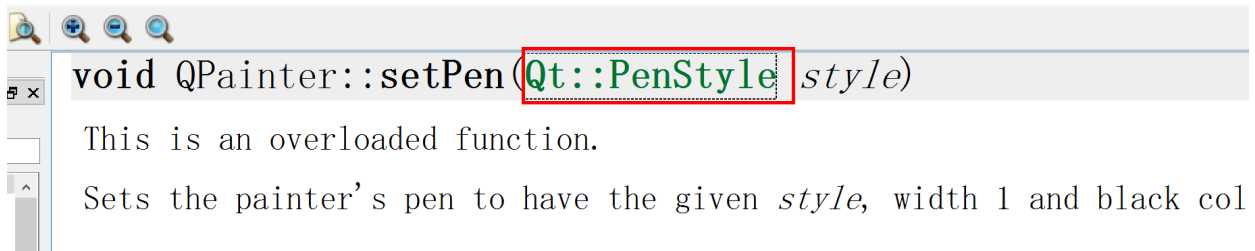
```
void QPainter::setPen(const QColor &color)
```

This is an overloaded function.

Sets the painter's pen to have style `Qt::SolidLine`, width 1 a

white	black	cyan	darkCyan
red	darkRed	magenta	darkMagenta
green	darkGreen	yellow	darkYellow
blue	darkBlue	gray	darkGray
lightGray			

设置样式:



```

1 //画矩形
2 //画家对 画笔进行设置
3 //painter->setPen(Qt::DotLine); //样式
4 painter->setPen(Qt::red); //颜色 红色
5 painter->drawRect(50,50,100,100);

```

运行结果:

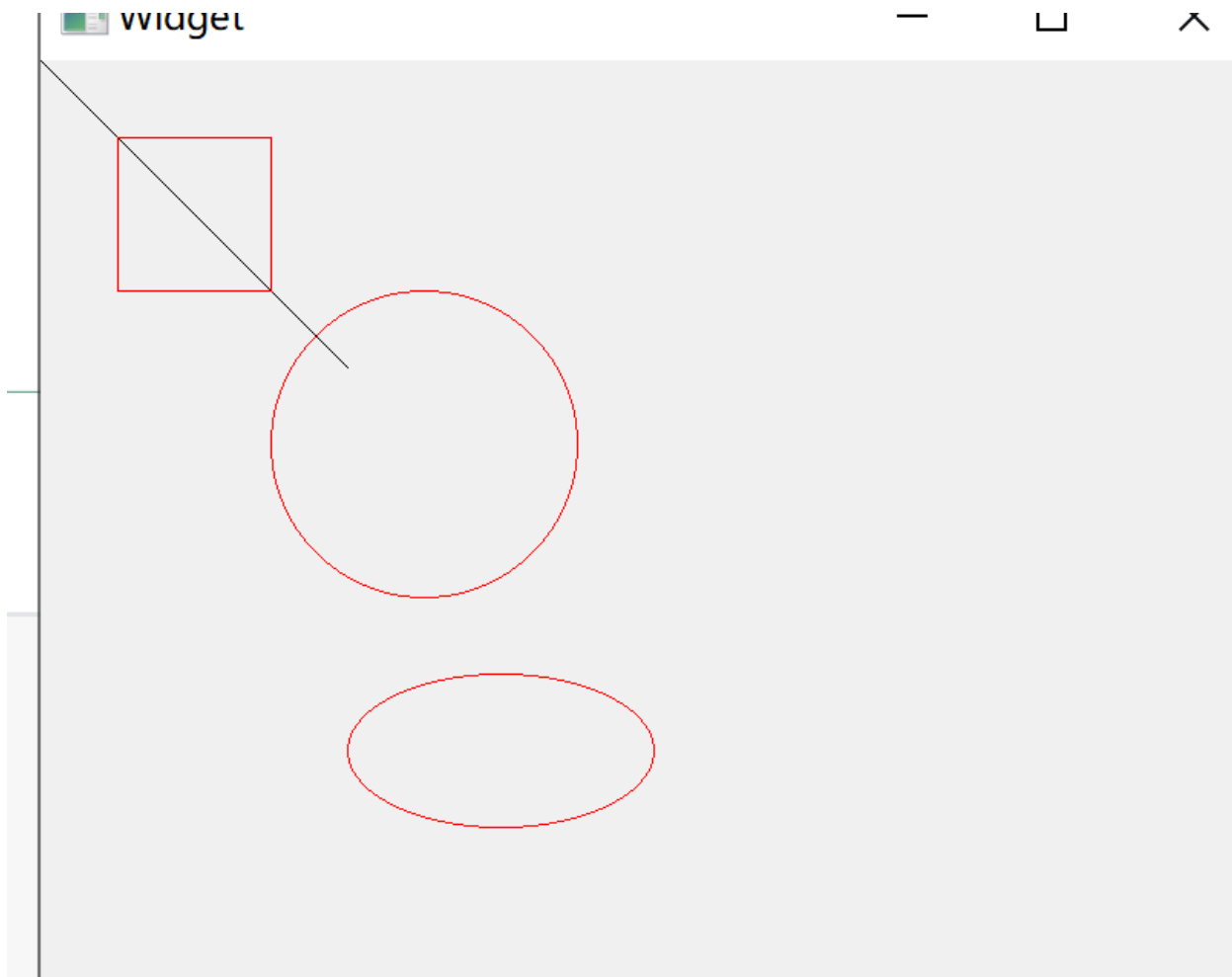


3、画圆

```
void drawEllipse(const QRectF &rectangle)
void drawEllipse(const QRect &rectangle)
void drawEllipse(int x, int y, int width, int height)
void drawEllipse(const QPointF &center, qreal rx, qreal ry)
void drawEllipse(const QPoint &center, int rx, int ry)
```

```
1 //画圆
2 painter->drawEllipse(150,150,200,200);
3 //画椭圆
4 painter->drawEllipse(200,400,200,100);
```

运行结果：



知识点3 【绘图设备】

是 [QPixmap](#)、[QBitmap](#)、[QImage](#) 和 [QPicture](#)。其中，

- [QPixmap](#) 专门为图像在屏幕上的显示做了优化
- [QBitmap](#) 是 [QPixmap](#) 的一个子类，它的色深限定为 1，可以使用 [QPixmap](#) 的 [isQBitmap\(\)](#) 函数来确定这个 [QPixmap](#) 是不是一个 [QBitmap](#)。
- [QImage](#) 专门为图像的像素级访问做了优化。
- [QPicture](#) 则可以记录和重现 [QPainter](#) 的各条命令。

案例1：QBitmap

在 `widget.cpp` 的构造函数中：

```

5  Widget::Widget(QWidget *parent) :
6      QWidget(parent),
7      ui(new Ui::Widget)
8  {
9      ui->setupUi(this);
10     this->resize(800,600);
11
12     //定义QBitmap一个绘图设备
13     QBitmap bit(200,200);
14     //定义一个画家
15     QPainter painter(&bit);
16     painter.drawEllipse(QPoint(100,100),100,100);
17     //保存图片
18     bit.save("C:\\work\\qt\\day21\\02_test\\bit01.jpg");
19 }
20
21 ~Widget()

```

运行结果：

此电脑 > BOOTCAMP (C:) > work > qt > day21 > 02_test

名称	修改日期	类型
02_test.pro	2020/4/10 10:59	Qt Pro
02_test.pro.user	2020/4/10 11:00	Per-Us
bit01.jpg	2020/4/10 11:35	JPG 图
main.cpp	2020/4/10 10:59	C++ S
widget.cpp	2020/4/10 11:35	C++ S
widget.h	2020/4/10 11:05	C++ H
widget.ui	2020/4/10 10:59	Qt UI

案例2：image的像素操作

在QImage类中：

```

void setOffset(const QPoint &offset)
void setPixel(const QPoint &position, uint index_or_rgb)
void setPixel(int x, int y, uint index_or_rgb)
void setPixelColor(const QPoint &position, const QColor &color)
void setPixelColor(int x, int y, const QColor &color)
void setText(const QString &key, const QString &text)
QSize size() const

```

在widget.cpp的构造函数中:

```
1  #if 1
2  //定义QImage一个绘图设备
3  QImage img;
4  img.load(":/image/up.png");//事先添加资源
5
6  for(int i=50;i<100;i++)
7  {
8      for(int j=50;j<100;j++)
9      {
10         int value= qRgb(255,0,0);
11         img.setPixel(i,j, value);
12     }
13 }
14
15 //定义一个画家
16 QPainter painter(&img);
17 painter.drawEllipse(QPoint(30,30),30,30);
18 //保存图片
19 img.save("C:\\work\\qt\\day21\\02_test\\imge02.jpg");
20 #endif
21
```

运行结果:

名称	修改日期
image	2020/4/10 11:40
02_test.pro	2020/4/10 11:41
02_test.pro.user	2020/4/10 11:00
bit01.jpg	2020/4/10 11:35
imge02.jpg	2020/4/10 11:49
main.cpp	2020/4/10 10:59
res.qrc	2020/4/10 11:41
widget.cpp	2020/4/10 11:49
widget.h	2020/4/10 11:05
widget.ui	2020/4/10 10:59



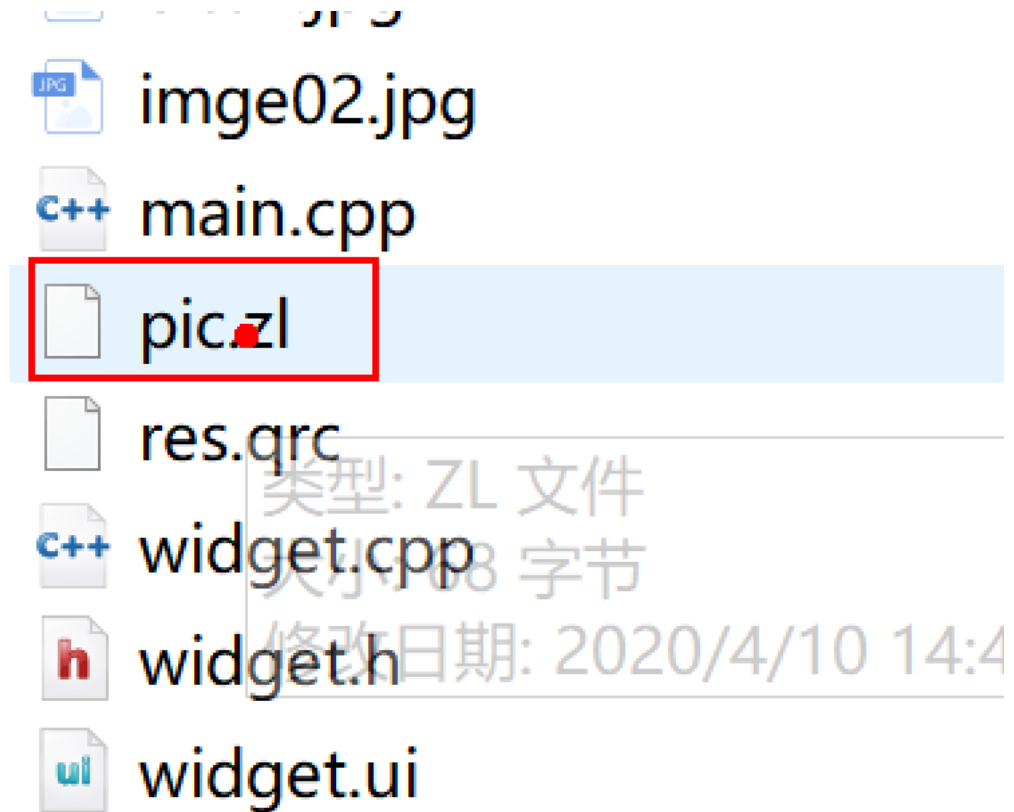
案例：重现绘图指令QPicture

在widget.cpp的构造函数中（保存绘图指令）

```
1  #if 1
2  //绘图设备
3  QPicture picture;
4  //画家
5  QPainter painter;
6
7  //记录绘图指令
8  painter.begin(&picture);
9  painter.drawEllipse(100,100,100,100);
10 //结束记录绘图指令
11 painter.end();
```

```
12
13 //保存绘图指令
14 picture.save("C:\\work\\qt\\day21\\02_test\\pic.zl");
15
16 #endif
```

运行结果：

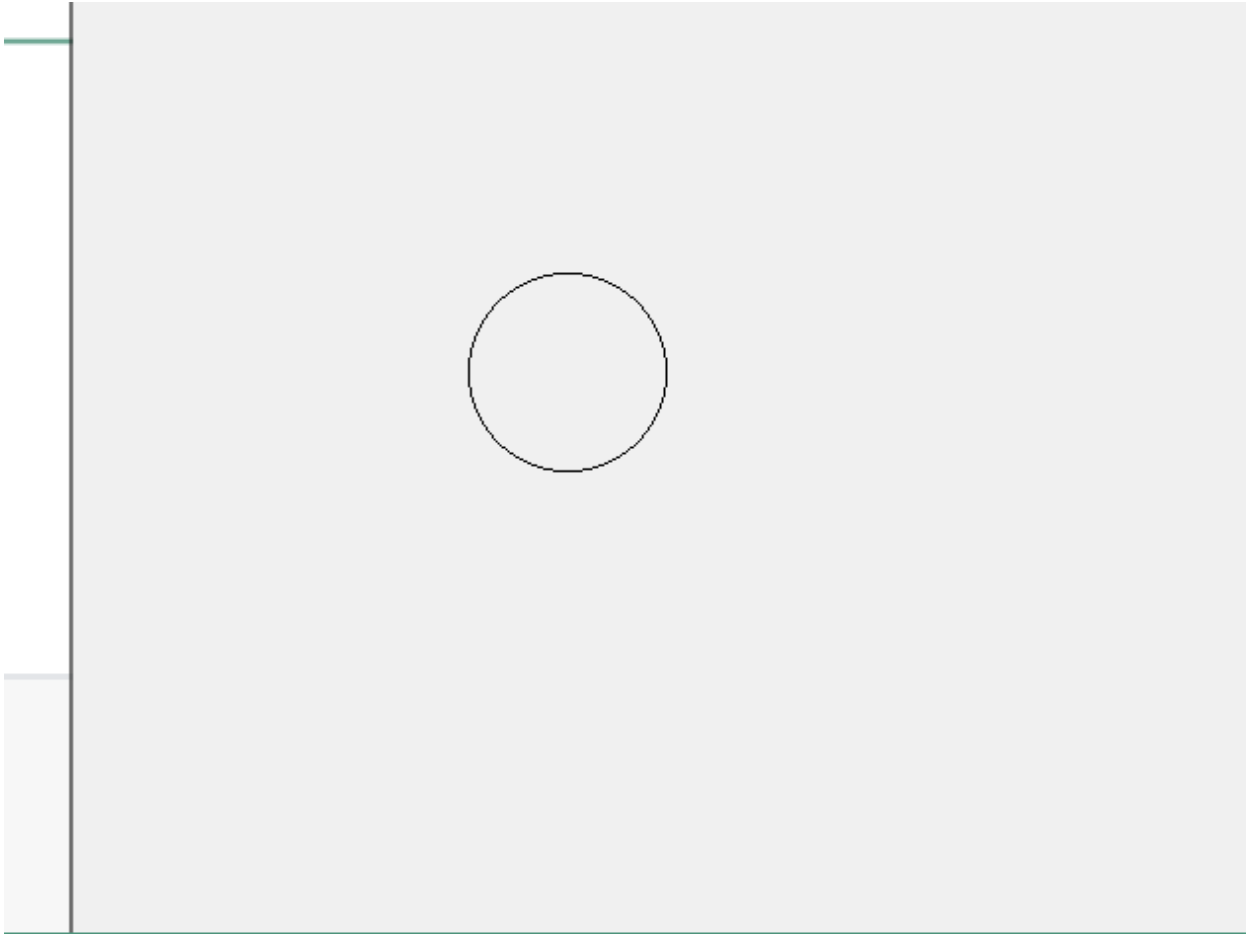


重现pic.zl 到主窗口中（绘图事件中完成）

在widget.cpp的绘图事件中写

```
1 #if 1
2 QPicture picture;
3 QPainter painter(this);
4 //绘图设备picture 加载绘图指令
5 picture.load("C:\\work\\qt\\day21\\02_test\\pic.zl");
6
7 //画家根据 绘图指令绘图
8 painter.drawPicture(100,100, picture);
9 #endif
```


运行结果：

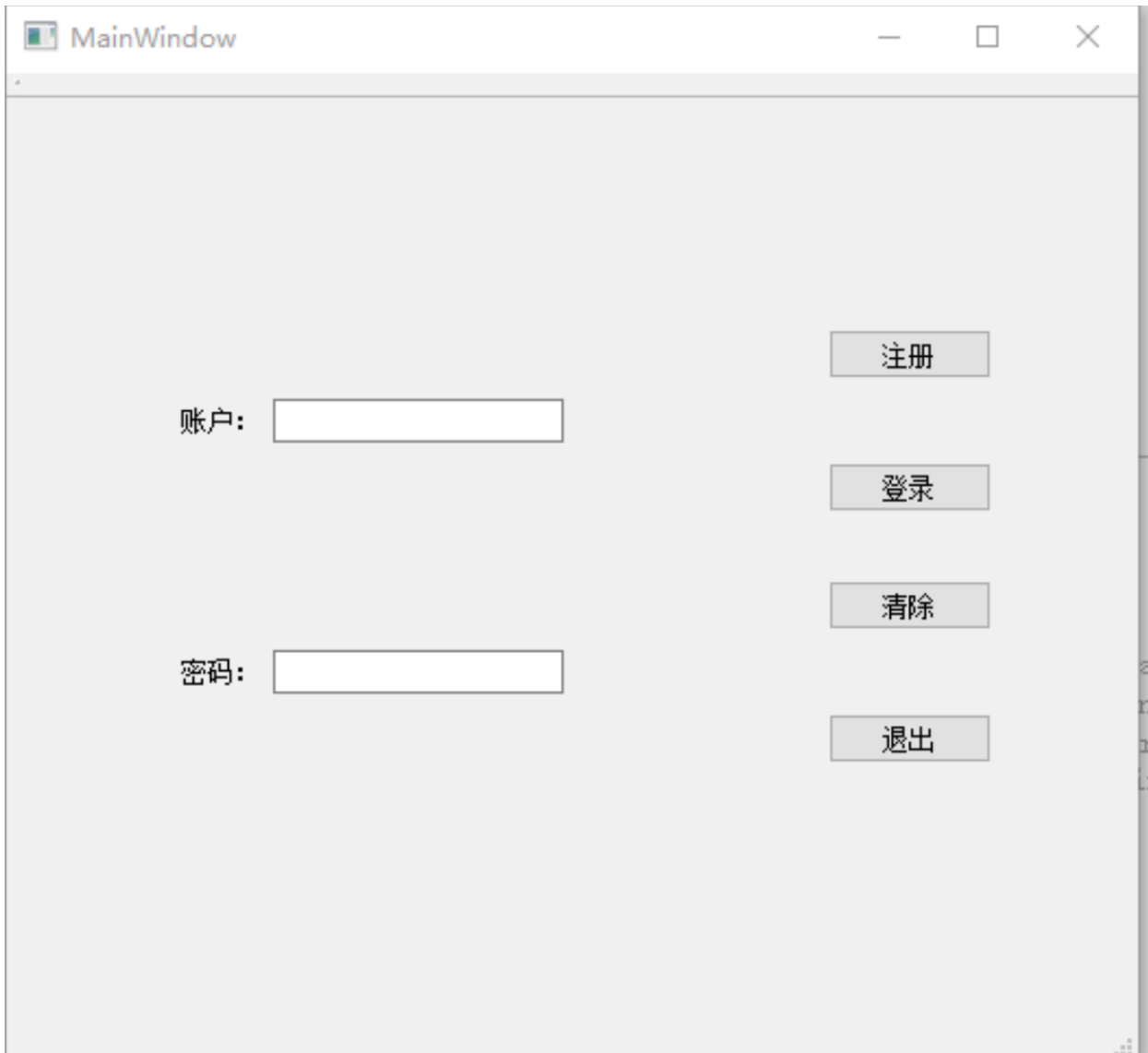


知识点4【项目篇】

每个人必须完成（工作日4天）

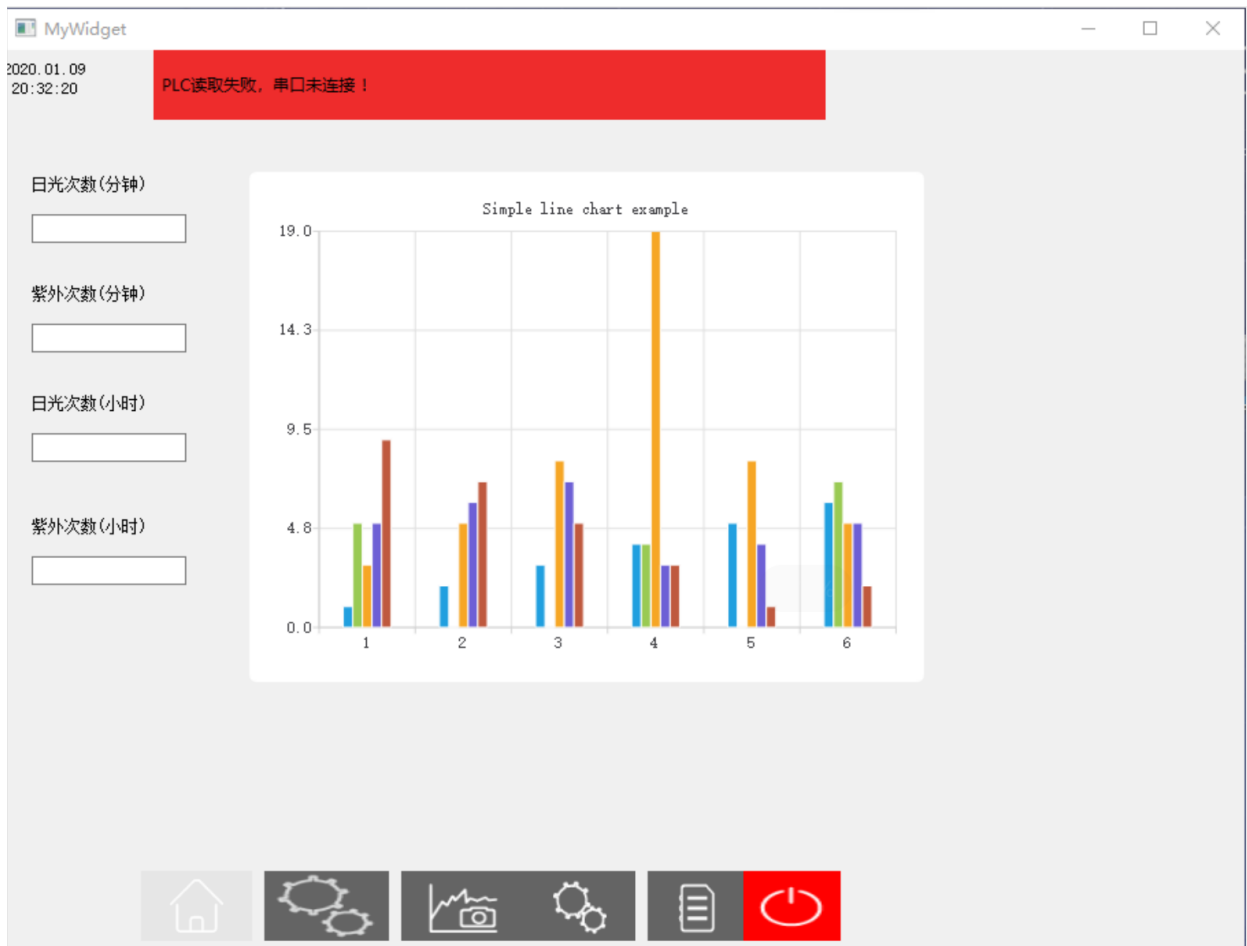
项目1：工业控制的UI（必须完成）

1、注册登录界面：

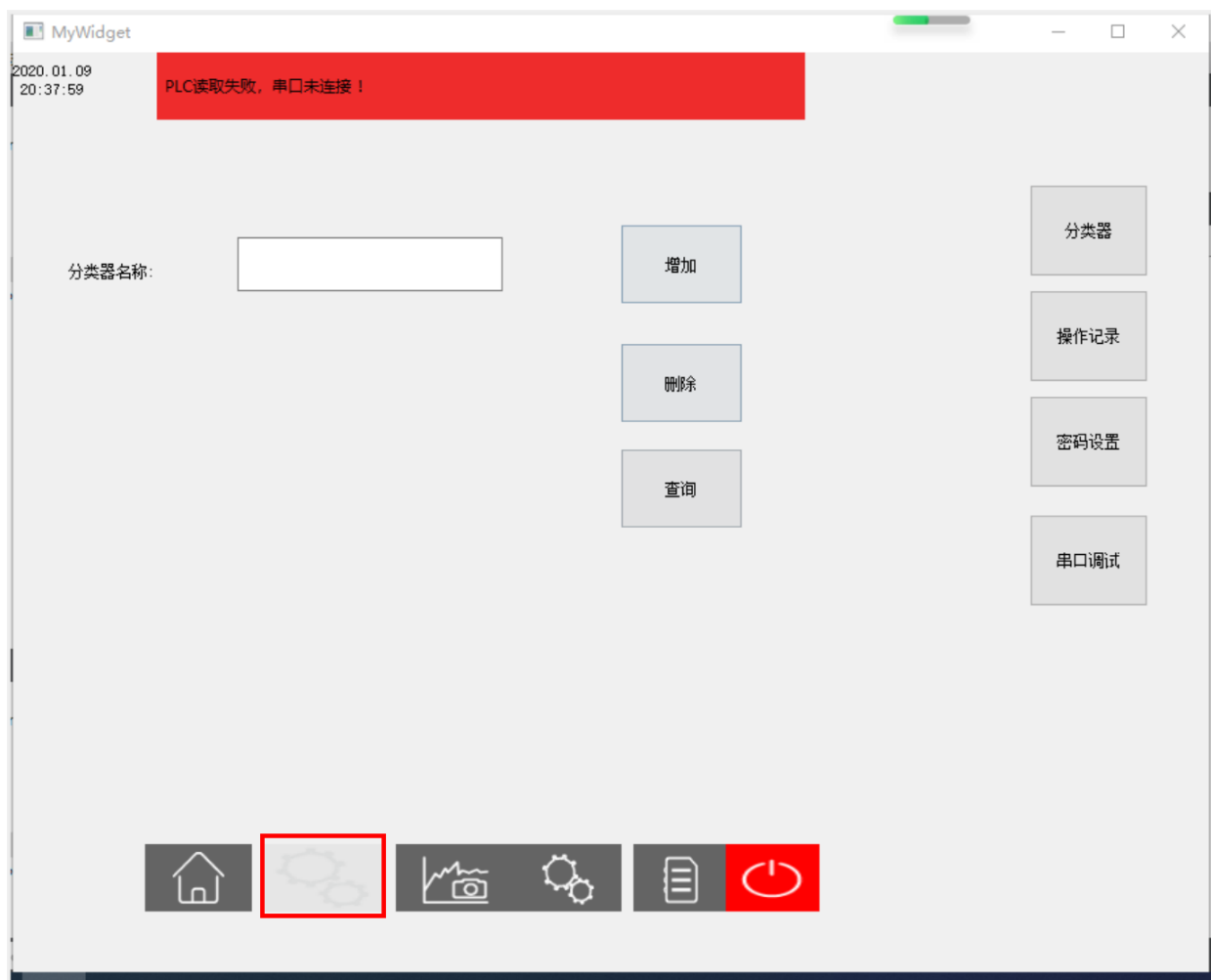


单击注册：跳到注册界面（用户名 密码 确认密码），将用户名 密码保存到文件中
输入账户 密码 如果单击的"登陆" 获取 账户 密码 从文件中查找 是否有该账户和密码 成功
跳入下一个页面 失败提示重新输入。

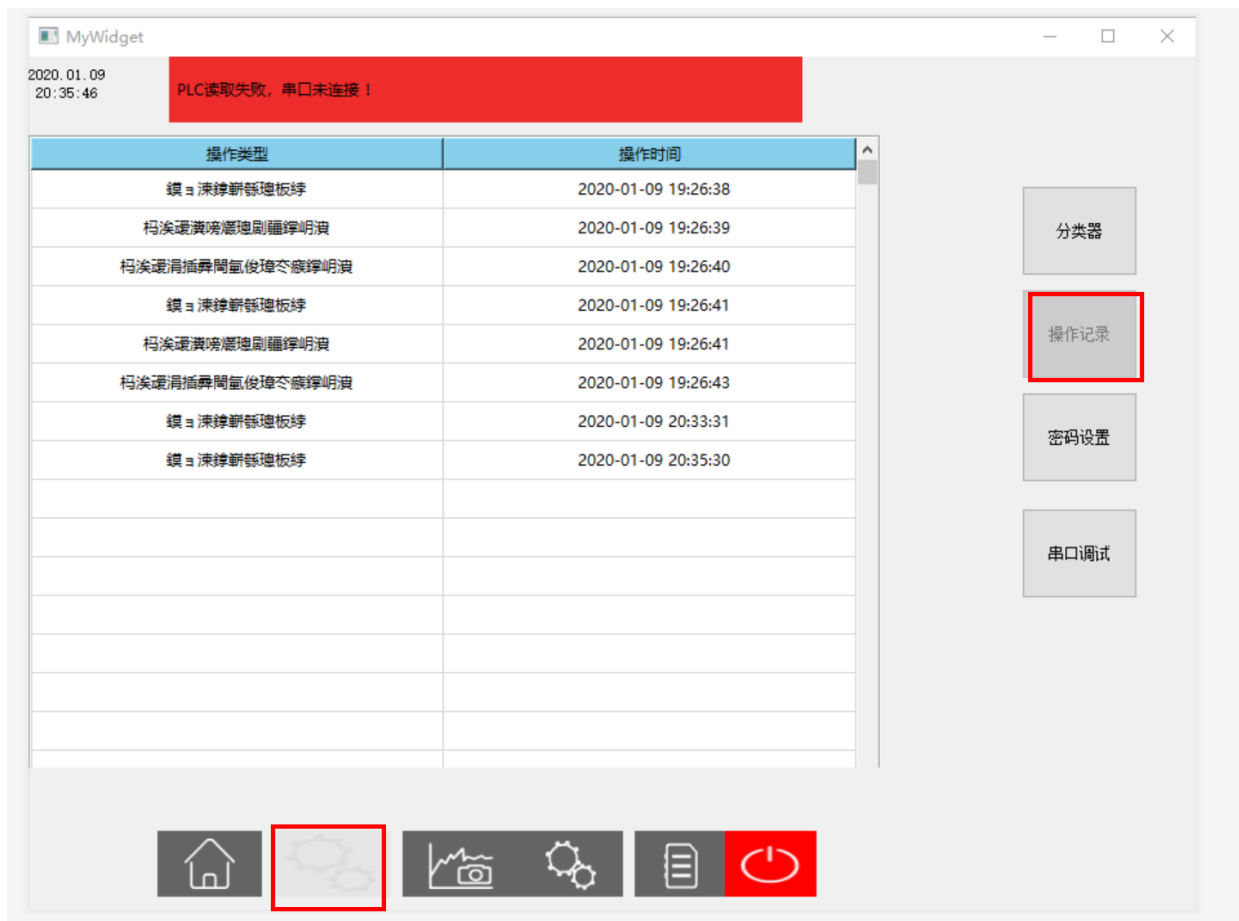
2、主页面 柱状图 创新



3-1设置页面



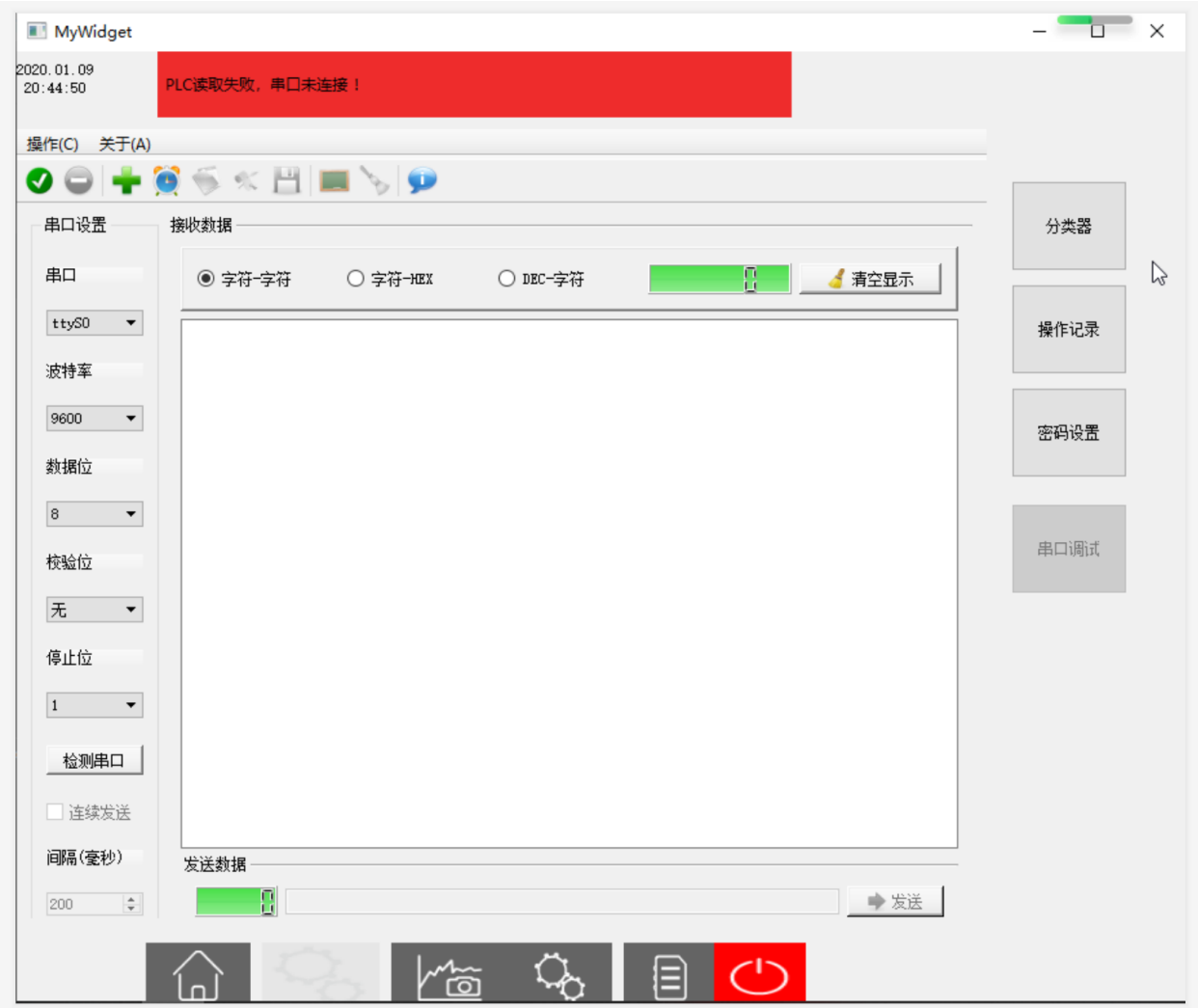
3-2: 操作历史记录



3-3: 密码设置



3-4：串口调试



波特率

9600 ▼

9600 ▲

14400

19200

38400

56000

57600

76800

115200

128000

256000 ▼

无 ▼

操作(C) 关于(A)



添加串口



打开串口



关闭串口



保存数据



读取文件



写入文件



读取间隔



计数清零



清空串口



退出程序



接收数据



字符-字符

串口

ttyS0

ttyS0

ttyS1

ttyS2

ttyUSB0

9600

关于(A)



关于Lincom

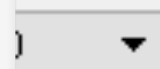


设置

接收数据

☒ 字符-字符

☐ 字符-HEX



128

数据位

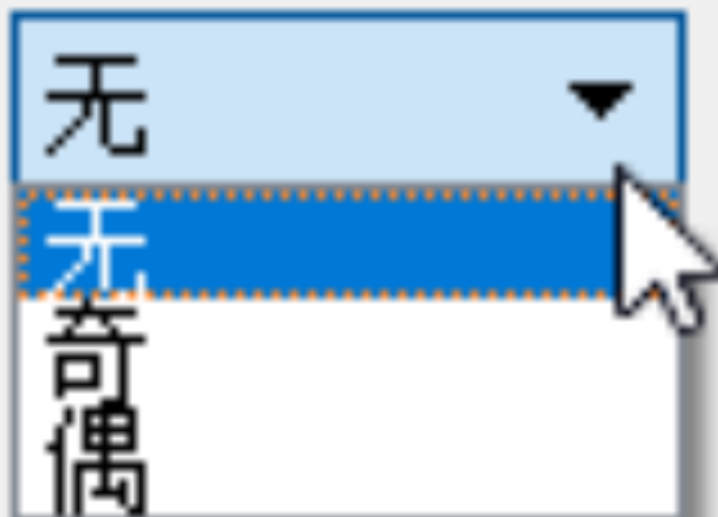
8	▼
5	
6	400%
7	
8	

无	▼
---	---

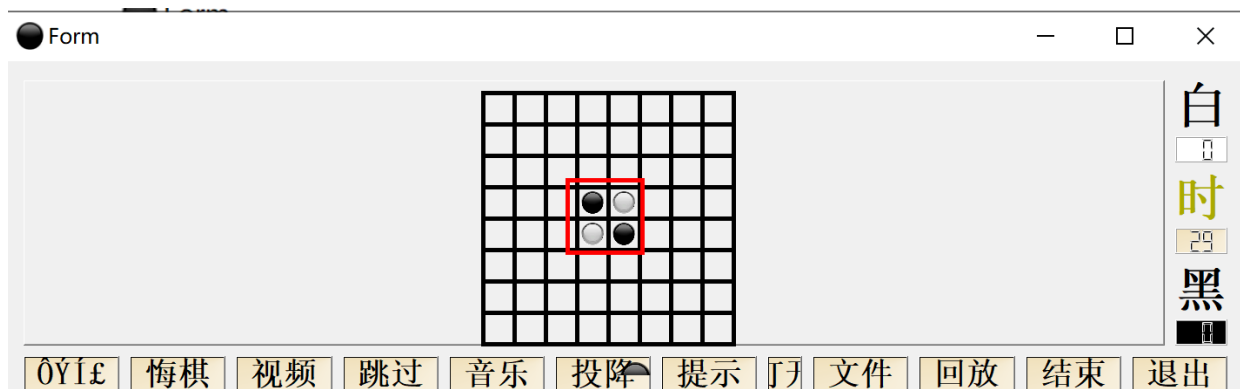
停止位

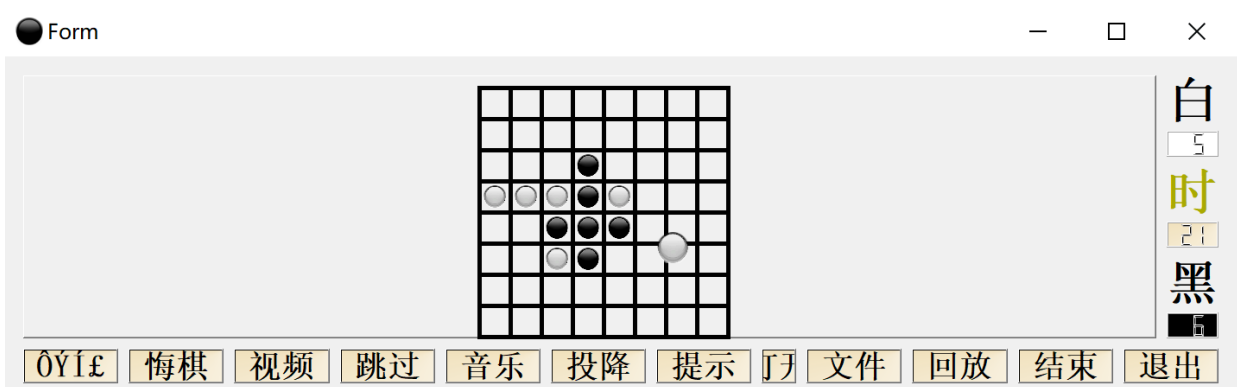
1
1
1.5
2

校验位



项目2：棋牌游戏(尽量完成+加分项)





1、规则介绍:

规则是在8*8的棋盘上的中心位置，先放入四个棋子，黑白各两个。默认是黑子先下，然后双方轮流下子，在直线或斜线方向，己方两子之间的所有敌子（不能包含空格）全部变为己子（称为吃子）。每次落子必须有吃子。双方都不能吃子的时候，即可判断胜负，此时子多者为胜

2、功能需求:

1) 基本功能:

A) 双人对战：通过人为点击鼠标实现黑白棋轮流下子

B) 人机对战：人落子后，必须要有1到2秒的延时时间，然后电脑开始下棋。

C) 悔棋：要求悔棋能悔到开始的位置，同时角色同步(本来开始是黑子先下的，悔棋悔到

开始位置的时候，变为白子先下，这样是不合理的)

D) 角色提示：当该黑子(或白子)下的时候，要求让用户能看到提示，知道该谁下子了。

E) 切换角色：有三种情况：1)当一方不能吃子的时候，切换为另一方的角色 2)当一方吃子 后，切换角色 3)倒计时时间到了，切换角色

F) 倒计时：要求有一个倒计时(通常为30s)，时间倒计为0s时，切换角色

G) 双方棋子个数提示：要求能让用户能看到当前黑白棋双方棋子个数为多少

H) 胜负判断：双方都不能吃子的时候，即可判断胜负，此时子多者为胜，要求能让用户看到胜负提示

1) 拓展创新:

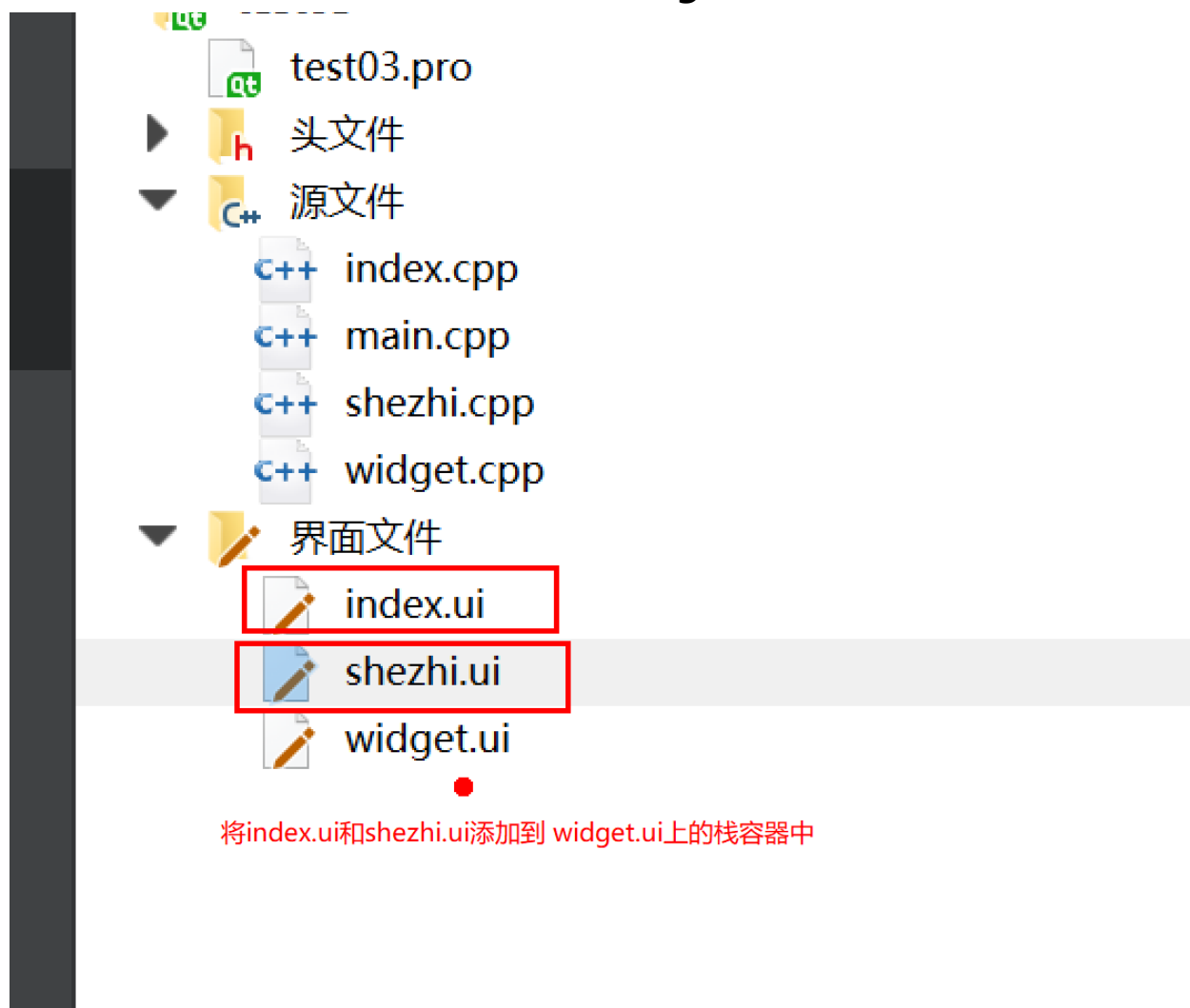
提示：黑白棋项目的创新在评分时创新的个数不封顶，创新越多，分数越高。创新的原则是：有新意，有技术含量，人机交互友好等。

下面只是例举了个别创新，大家可以无限拓展自己的思维。

- A) 网络对战：可以在局域网类对战
- B) 聊天功能：可以边下棋边聊天
- C) 存档读档：可以把棋盘状态保存起来，重开软件后，可以恢复这个棋盘的状态，并且能 接着下
- D) 播放视频：注意是播放视频，播放gif动画
- E) 提示所有能下子的位置：注意，能够提示所有能下子的位子才能算一个创新
- F) 托管和提示（合并为一个创新）：到当能够托管让机器帮我们下子，还有，能提示下子的 位置(只提示一个位置，不是所有位置)，这两个合为一个创新
- G) 切换背景、切换按钮图标、切换鼠标图案(合为一个创新)
- H) 布局：界面做得挺好看，并且布局了

项目一：补充知识点

1、窗口内切换 多个窗口 (QStackWidget)



改变样式表...

大小限定

提升为...

转到槽...


2 的页 1

插入页

下一页

上一页

改变页顺序...

 放到后面(B)

删除

提升为...

总页数

当前页数

添加新的也

切换页码