**QWiget：**

窗口基类有三种选择：

QWidget(空窗口)是QMainWIndow(菜单栏工具，状态栏)和QDialog(对话框)的基类

Generate form创建UI界面

第一个程序

QApplication a(argc, argv); a应用程序对象，在qt中，应用程序对象有且仅有一个

myWidget w窗口对象 myWidget父类 ->QWidget w是myWidget派生类实例化的对象

return a.exec();让应用程序对象进入消息循环，当点击×后退出循环

**命名规范**： 类名：首字母大写 函数名 变量名：首字母小写，单词单词之间大写

**快捷键：**

注释 ctrl+/ 运行 ctrl +r 编译 ctrl +b 缩放 ctrl+鼠标滚轮

查找ctrl +f 整行移动 ctrl +shift +↑或↓ 帮助文档 F1 自动对齐 ctrl+i

同名之间.h和.cpp切换 F4

**创建按钮：**

QPushButton\*btn=new QPushButton;

btn->setParent(this);使button成为该类的一个部件，所有Widget的派生类都可以使用这个函数

btn->setText("the first button.");

**或者(按照控件的大小创建窗口)**

QPushButton\*btn=new QPushButton(“the second button”,this);

**设置大小：**

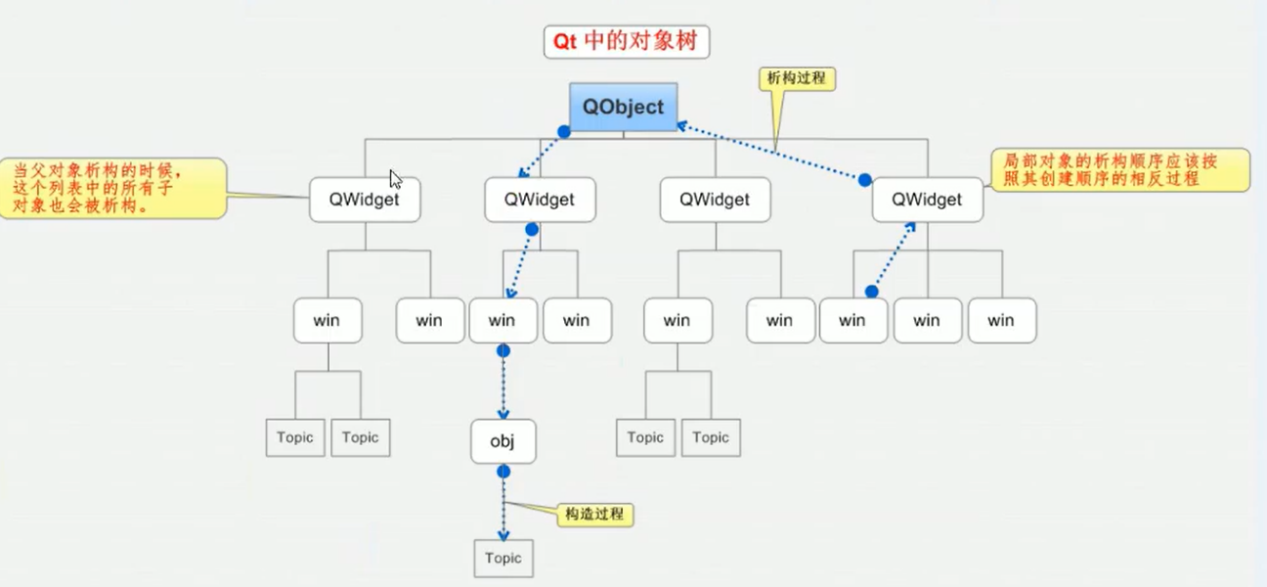
resize(weight,height)重置窗口大小

同理可以使用btn->resize(weight,height)来重置按钮的大小

setFixedSize(weight,height)固定窗口大小

**设置窗口标题：**

setWindowTitle(“”);

**对象树的概念：**

QObject以及其派生类析构时会自动释放将其指定为parent的在堆区开辟的内存。

**链接信号和功能：**

原型：QMetaObject::Connection QObject::connect(const QObject \*sender, const char \*signal, const QObject \*receiver, const char \*method, Qt::ConnectionType type = Qt::AutoConnection)

实例：connect(btn2,&QPushButton::clicked,this,&QWidget::close);**函数指针链接（函数地址）**

connect(btn2,SIGNAL(clicked()),this,SLOT(close()));**旧版本**

信号可以在成员成员之间传播，也可以在成员和槽之间传播

**输出到控制台：**

直接使用qDebug()使用类型cprintf()

包含头文件#include<QtDebug>可以使qDebug()<<（like cout）

**如何获得类内非静态函数的地址? &类名::函数名，没有括号。**

**信号与槽：**

信号写在signal下，槽写在public或全局函数下(过去写在public plot)

信号的释放：Qt5以上自定义槽函数写在相同于函数的语法。过去：emit 函数名()；

信号的参数可以传递到槽，二者直接的流动性体现出来

**自定义槽函数重载问题（函数指针的使用）：**

问题出现：有两套信号和槽如果直接使用通过取址无法识别为哪一个信号和槽。

解决：使用函数指针（在指针前要加上作用域确定函数的位置）

void(Student:: \*stsignal)(QString)=&Student::sendsignal;

char\*debug时不会加””QString会加上

如何转化QString至char\*? .toUTF8()转化为QByteArray再.data()转化为char\*

qDebug()<<"thats second debugs"<<name.toUtf8().data();

**断开连接：**

disconnected()

将按钮连接信号？

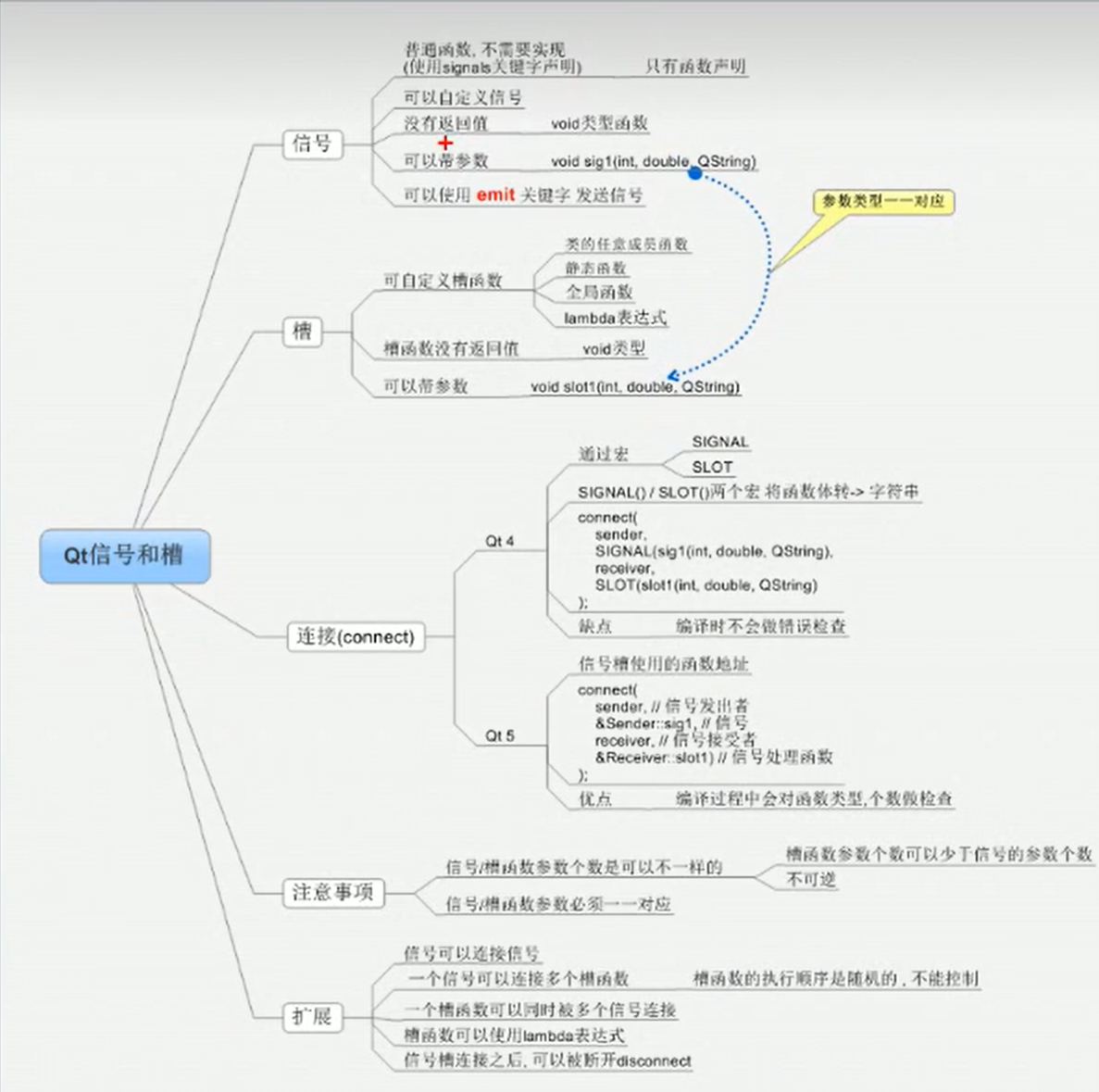
一个信号可以连接多个槽，多个信号可以连接一个槽。

槽函数和信号的类型要匹配，个数信号可以多余槽

**lambda表达式：[捕获列表](参数列表) -> 返回类型 {函数体}**

[&/=/b/=,&b](传参){ btn->setText("第二次发送信号"); }()mutable关键字可加;最后的括号表示调用函数

使用lambda表达式可以调用想调用的函数不用考虑参数匹配的问题



**04homework中：**

子窗口若没有设置为独立窗口，会覆盖在父窗口上，导致父窗口上的被覆盖区域功能无法实现，需要使用函数setWindowFlags(Qt::Window); // 设置窗口标志为独立窗口

**QmainWindow：**

**菜单栏：**

QMenuBar \*Bar =new QMenuBar();菜单栏创建（也可以使用便捷函数QMenuBar \*Bar = menuBar();）

setMenuBar(Bar); 将菜单栏放入窗口

QMenu \*fileMenu = Bar->addMenu("文件");QMenu \* editMenu= Bar->addMenu("编辑");创建菜单

两种方式添加菜单功能：

QAction \*openAction = new QAction("打开", this);

fileMenu->addAction(openAction); //这种将每一个功能看成一个对象，更为常用

fileMenu->addAction("新建");//便捷函数，无法链接信号槽

fileMenu->addSeparator()添加分割线;

**工具栏**

QToolBar \*toolbar=new QToolBar(this);创建工具栏

addToolBar(Qt::RightToolBarArea,toolbar);往特定区域添加工具栏

QAction\*ChangeWordSize=new QAction("大小",this);添加功能

QPushButton \*btn=new QPushButton("设置",this);

toolbar->addWidget(btn);//添加按钮配件

toolbar->addSeparator();添加分割线

toolbar->setAllowedAreas(Qt::LeftToolBarArea|Qt::RightToolBarArea);设置停靠区域

toolbar->setFloatable(false);设置是否浮动

toolbar->setMovable(false); 设置移动（设置为false后以上都没意义）

**状态栏**

QStatusBar\*statusbar=new QStatusBar(this);创建状态栏

setStatusBar(statusbar);

QLabel \*label=new QLabel("字数",this);

statusbar->addWidget(label); //放置标签控件

**铆接部件(浮动窗口)可以有多个**

QDockWidget\*dockwidget=new QDockWidget("浮动窗口",this);

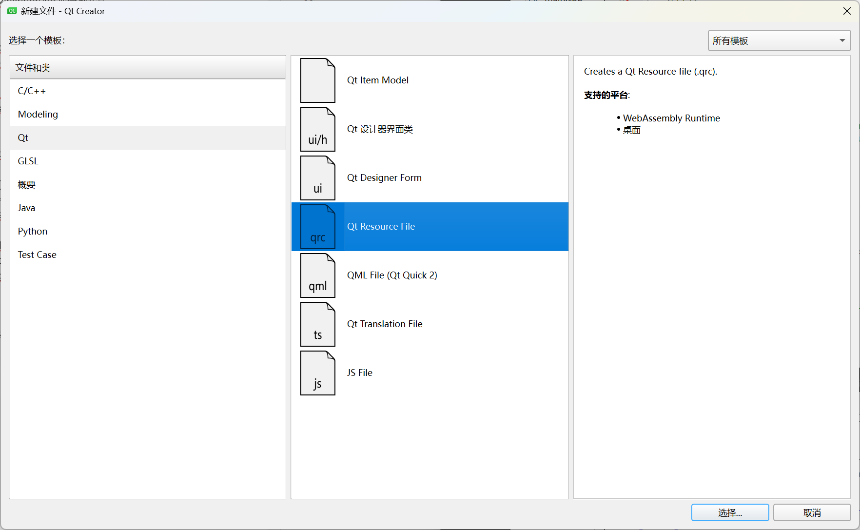
addDockWidget(Qt::BottomDockWidgetArea,dockwidget);给特定区域浮动窗口

**核心部分(以textedit为例子)**

QTextEdit \*edit=new QTextEdit(this);

setCentralWidget(edit);//设置中心部件

**添加资源文件**

ui->actionnew->setIcon(QIcon("C:/Users/17677/Pictures/Screenshots/p2.png"));以绝对路径直接添加文件

使用添加Qt资源“: + 前缀名 + 文件名”

右键项目中文件头点击添加新文件获得资源文件夹，将需要的资源文件放在项目目录下。

右键.qrc文件open in editor，添加前缀和添加文件。

ui->actionnew->setIcon(QIcon(":/new/prefix1/p1.png"));

其中antionnew源自于action+菜单名

seticon设置图标，ui设计中的操作。

**添加自定义对话框（包含头文件QDialog）**

对话框分类

模态对话框(不关闭就不可以对其他窗口操作)

QDialog dlg(this)创建对象并制定基类

dig.resize(200,100)设置大小 dlg.exec();启动应用事件循环

非模态对话框(可以对其他窗口操作)

QDialog \*dlg2=new QDialog(this); 创建对象

dig.resize(200,100)设置大小dlg->show();展示

**区别：exec() 和 show()**

**如何解决创建对话框在整个程序关闭后才统一释放？**

dlg1->setAttribute()设置属性

void QWidget::setAttribute(Qt::WidgetAttribute attribute, bool on = true)第一个参数是属性的枚举值，第二个参数是决定属性是否运用。

**Qt标准对话框（常用）：**

**QMessageBox显示一条消息并提供一个或多个按钮供用户选择：**

使用静态成员函数可以快速创建和显示系统自带的对话框，无需显示的创建对象

question的默认参数略有不同于剩余3个，其他地方相同，都是1.父类 2.框上的文本 3.中间的文本 4.按钮的种类 5.默认按钮

**实例：**

QMessageBox::critical(this,"critical"emmm...",QMessageBox::Apply|QMessageBox::Close,QMessageBox::Close);

QMessageBox::information(this,"information","请缴纳费用");//后面可以填button两个参数(前两个和最后一个静态函数的默认参数列表一样)

QMessageBox::question(this,"question","疑问？");

QMessageBox::warning(this,”warning”,”警告”);

**作用：利用if else语句在代码块内执行相应操作**

if(QMessageBox::Apply==QMessageBox::critical(this,"critical","emm...",QMessageBox::Apply|QMessageBox::Close,QMessageBox::Close)){qDebug()<<"缴纳中";}

else{

qDebug()<<"Warning...";

}

创建对象，使用成员函数可以自定义操作满足特定需要。

**QFIleDialog用于用户选择文件或目录**。

QString str=QFileDialog::getOpenFileName(this,"打开文件” ,"C:/Users/17677/Desktop",

"(\*.txt)");返回值是绝对路径

**QInputDialog**允许用户输入文本或数字。(获取用户输入的信息)

QInputDialog::getText/getInt/getDouble()用相应的类型接收即可

QString S=QInputDialog::getText(父类,"标题","提示标签文本",输入模式(QLineEdit::Normal：正常显示文本, QLineEdit::NoEcho：不显示输入的字符, QLineEdit::Password：显示输入的字符，但用“•”替代) ,"默认文本",nullptr(接收用户是否点击了确定按钮的bool指针),对话框的模式(默认为模态对话框)),; 使用S进行接收

**QColorDialog**用于选择颜色，例如设置文本颜色、背景颜色等。

QColor(255,0,0)(r,g,b,透明度);

QColor c=QColorDialog::getColor(QColor(255,0,0));//返回值是一个QColor

QFontDialog 用于选择文本的字体和样式。

QProgressDialog显示任务的进度，并可选择取消任务。

**布局**

widget进行水平竖直网格操作

加入弹簧控制左右上下间隙

layoutLeftMargin调整窗口和控件之间距离

echomode调整输入模式

设置垂直策略水平策略

设置最大窗口和最小窗口来限制拉伸。

**按钮**

**Tool button可以包含图标或者文本，可以带有下拉箭头，提供多种工具操作。**

toolbuttonstyle 设置图片和文字的关系 autoraise凸起风格

**(单选)Radio button提供选项**，结合**group box**使用；可以使用setchecked()选择默认选中的人。

**(多选)Check box，**结合**group box**使用；

**Qwidget容器**

QStringList对象可以直接使用<<进行拼接操作，而QStringList指针则需要->append成员函数来进行拼接操作。

item Views（基于数据库操作）

**item Widgets（基于项目自身）**

**QListWidget：**

**法一：**

包含头文件#include<QListWidgetItem>其对象控制控件中单行或单个项目

使用additem将widgetitem加入到list中

for循环进行垂直居中，利用了listwidget->count()获取对象数，listWidget->item(i)获取第i个对象，setTextAlignment(Qt::AlignCenter)设置对齐格式。

例子：

QString test="嗯嗯你说的对";

QListWidgetItem\* item1=new QListWidgetItem(test);

QListWidgetItem\*item2=new QListWidgetItem("呃呃，没绷住");

ui->listWidget->addItem(item1);

ui->listWidget->addItem(item2);

for(int i=0;i<ui->listWidget->count();i++){//使用for循环使每一行居中

QListWidgetItem\*item=ui->listWidget->item(i);

item->setTextAlignment(Qt::AlignCenter);

}

**法二：（使用字符串链表，可以直接使用<<进行拼接）**

创建QStringList对象，list<<”输入内容”<<”…”;

ui🡪listwidget🡪addItems(QStringList);

例子：

QStringList list;

list<<"emm呃呃呃"<<"没话说了";

ui->listWidget->addItems(list);

**QTreeWidget：**

ui->treeWidget->setHeaderLabels(QStringList()<<"菜系"<<"制作方法");创建标签

QTreeWidgetItem \*item1=new QTreeWidgetItem(QStringList()<<"")创建顶层节点或子节点

父节点->addchild(子节点)将子节点加到父节点上。

**QTableWidget：**

需要设置列数和设置行数

ui->tableWidget->setColumnCount(3); ui->tableWidget->setRowCount(5);

设置水平表头

ui->tableWidget->setHorizontalHeaderLabels(QStringList()<<"姓名"<<"年龄"<<"ID");

可以创建QStringList或者QStringList\*来储存数据，使用for循环加入到

ui->tableWidget->setItem(i,j++,new QTableWidgetItem(namelist[i]));

**QStackWidget：**

可以创建多个界面进行切换。

ui->stackedWidget->setCurrentIndex(0); 设置默认页面

connect(ui->pushButtontext1,&QPushButton::clicked,ui->stackedWidget,[=](){

ui->stackedWidget->setCurrentIndex(1);

});建立按钮和界面的链接

**Comboboxc：**

ui->comboBox->addItem(QString);或者ui->comboBox->addItems(QStringList);添加

**使用label输出gif和png**

现将图片和gif添加在资源文件中(需要在资源文件内才能被找到)

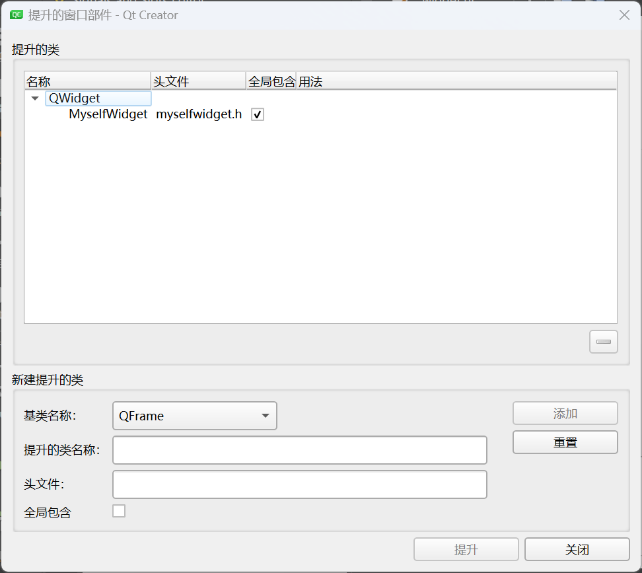
ui->labelpic1->setPixmap(QPixmap(":/new/1/p2.png"));利用了临时对象和构造函数创建了一个QPixmap对象

QMovie\*mov=new QMovie(":/new/1/th\_pig.gif");

ui->labelgif->setMovie(mov);需要传入QMovie \*

mov->start();播放动图

ui->(对象名)->setScaledContents(true);控件自适应大小。



**创建自定义控件：**



创建一个带有ui界面的类。

在需要添加该控件的ui界面，创建一个widget，右键fliter，将该widget提升为自定义类

**QEvent：**

**鼠标信息：#include<QMouseEvent>**

鼠标进入函数是一个虚函数可以进行重写。

注意将将控件提升为自定义控件时需要类型相同。

因为QLabel是QWidget的子类，所以可以将文件中Widget改为QLabel即可修改自定义控件的属性。

**格式化字符串：**

QString s=QString(“%1 %2 %3”),arg().arg().arg();

**获取坐标：**

ev->pos()返回值是Point类

ev->position().x()/y()控件坐标 或者ev->globalposition.x()/y()全局坐标

**获取鼠标行为：**

ev->button()/buttons() 中有许多枚举值button处理触发鼠标单个事件（瞬间），buttons用于确定当前按键的组合的状态（持续时间的状态值）。

ev->buttons& Qt::LeftButton只要按下了左键就为True

ev->buttons== Qt::LeftButton只按下左键才为True

SetMouseTracking可以添加在构造函数中也可以按需添加。

**计时器（事件实现）：**

**工作原理：starttimer(周期)每一次周期后都会触发一次timerEvent()函数，执行语句块的内容后等待下一个周期。除非出现killtimer**

void timerEvent(QTimerEvent\*ev);重写时间来创建定时器。

startTimer(1000)启动定时器，单位为毫秒，返回值为这个定时器的Id

可以利用ev中的timerid进行比较

killTimer(id1)停止id1计时器

void Widget::timerEvent(QTimerEvent\*){

If(ev->timerId()==id1){//可以在头文件中给每一个定时器创建Id。

static int num=1;//静态变量的使用。

ui->label\_2->setText(QString::number(num++));//将数字转换为一个字符串

}每一个id与每一个starttimer对应

}

**计时器（类实现QTimer）：**

构造函数中创建对象QTimer\*timer=new QTimer(this);可以填写参数设置父亲

timer->start(500);其中一个重载，设置间隔 timer->start()启动定时器

Qtimer::timeout间隔时间到传出信号。

connect(timer,&QTimer::timeout,[=](){//利用信号和槽传递

static int num=0;

ui->label\_3->setText(QString::number(++num));

});

**event事件分发器**

bool event(QEvent\*v);

利用event事件分发器拦截事件（包括鼠标键盘等等操作），如果返回值是true就不向下分发事件了。

Qevent中的type枚举类型有着各种事件。

每个事件发生都会执行一遍event函数，如果返回值是true则不继续进行。

**聚焦策略：**

strongfocus：只有这样才能捕获到键盘

tabfocus：按下tab获取焦点。

clickfocus：点击获得焦点

wheelfocus：鼠标滚轮获得焦点。

nofucos：无焦点

**在事件发生器前可以给配件安装时间过滤器：**

安装了过滤器后，就会监视和拦截该控件的事件

1.ui->label->installEventFilter(this);给控件添加一个过滤器

2.重写过滤器事件

bool Widget::eventFilter(QObject\*obj,QEvent\*ev){

if(obj==ui->label){//判断哪个控件使用了过滤器

if(ev->type()==QEvent::KeyPress){

qDebug()<<"过滤器过滤走了键盘输入";

return true;

}

}

//其他的正常返回

return QWidget::eventFilter(obj,ev);

}

**QPainter：**

void Widget::paintEvent(QPaintEvent\*pt) 首先创建画图事件(内部代码自动执行)

QPainter painter(this);创建画家，参数为QPaintDevice \*device

QPen pen(QColor(255,0,0));设置画笔(参数不知道填什么可以多使用构造函数)

QBrush brush(Qt::cyan); 对于填充图形，可以使用画刷来填充颜色可以选择自带的枚举值

画笔的操作：

pen.setWidth(4); 设置画笔宽度

pen.setStyle(Qt::DashDotDotLine); 设置画笔风格

painter.setPen(pen); 让画家使用画笔(需要让画家使用画笔后才会起作用)

画刷的操作

brush.setStyle(Qt::DiagCrossPattern);设置画刷风格

painter.setBrush(brush); 让画家使用画刷

画家的操作：

painter.drawLine(QPoint(100,100),QPoint(300,400));划线操作

painter.drawEllipse(QPoint(100,100),100,100);画圆和椭圆

painter.drawRect(QRect(QPoint(100,100),QPoint(200,400)));画矩形

painter.drawText(QPoint(100,100),"好好好，好好好哦啊哦");写文字

高级设置

painter.setRenderHint(QPainter::Antialiasing);设置抗锯齿

painter.translate(100,0);移动画家位置

painter.save();保存画家状态

painter.restore();还原到最近一次保存的状态

画家的画图操作

painter.drawPixmap(QPoint(200,200),QPixmap(":/mypicture/p1.png"),QRect(QPoint(0,0),QSize(400,200)));规定画图的地点和需要画的部分。

painter.drawPixmap(QPoint(200,200),QPixmap(":/mypicture/p1.png"));

使用update只重新调用paintEvent事件

pixmap = pixmap.scaled(size, Qt::KeepAspectRatio);利用scaled函数将图片进行缩放。

**QPixmap和QImage和QPicture：**

三者都用于处理图像数据，可以加载、保存、显示和处理图像

QImage：图像数据以二维数组的形式存储，可以直接访问和修改像素级别的数据。用于图像处理和计算机视觉任务，如图像滤波、特征检测、物体识别等。

适用于需要直接访问和操作像素级别数据的场景，如图像处理算法的实现。

QPixmap：主要用于界面开发，用于在 Qt 应用程序中显示图像。

适用于需要在界面上显示图像的场景，如在用户界面中显示图标、图片、背景等。

QPicture：是用于记录和重放图形操作的类。它可以记录绘图操作序列，然后在需要时重放这些操作，以实现图形的重绘。

具体操作见17-18-19.

**QFile：**

QFile file(path);创建QFile对象处理文件

file.open(QIODeviceBase::);设置打开方式，可在Qtassitant中查询

QByteArray bay=file.readAll()/file.readline()读取全部数据或者一行数据利用QByteArray储存。利用readline会有一个隐式指针的使用配合atEnd实现功能。

ui->textEdit->setText(bay);将读取数据放置编辑框内。

file.close();最后关闭文件。

**QT转码问题：**在Qt中，默认支持的输出格式是utf-8，Qt6中舍弃了QTextCodec类对与编码的转换所以需要以QString作为中介。

1.将字节数组转换成QString类：fromLocal8Bit/fromUtf8对于不同字节数组要使用不同的转换方式。

2.将QString再转换成对应的字节数组：toUtf8()/toLocal8Bit()。

对gbk文件追加操作案例：

file.open(QIODevice::Append);

file.write(QString::fromUtf8("好好好，呃呃呃").toLocal8Bit());

file.close();

**QFileInfo：**

常用成员函数

suffix()QString获取后缀 path()QString文件路径 filename()QString文件名

filePath()QString获取文件(包括文件名) birthtime()QDataTime文件创建日期

lastModefied()QDataTime最后修改日期 size()Int文件大小

将QDataTime转换成QString的方法：QDataTime的成员函数.tostring(常用"yyyy/mm/dd hh:mm:ss")

具体见QtAssitant QDataTime中toString(const QString &format, QCalendar cal) const。

注：信号是流通的，子控件和主控件直接的联系也是依靠信号才能流通。