1. 自定义控件封装
   1. 添加新文件 - Qt – 设计师界面类 (.h .cpp .ui)
   2. .ui中 设计 QSpinBox和QSlider 两个控件
   3. Widget中使用自定义控件，拖拽一个Widget，点击提升为，点击添加，点击提升
   4. 实现功能，改变数字，滑动条跟着移动 ，信号槽监听。
   5. 提供 getNum 和 setNum对外接口
   6. 测试接口
2. Qt中的事件
   1. 鼠标事件
   2. 鼠标进入事件 enterEvent
   3. 鼠标离开事件 leaveEvent
   4. 鼠标按下 mousePressEvent ( QMouseEvent ev)
   5. 鼠标释放 mouseReleaseEvent
   6. 鼠标移动 mouseMoveEvent
   7. ev->x() x坐标 ev->y() y坐标
   8. ev->button() 可以判断所有按键 Qt::LeftButton Qt::RightButton
   9. ev->buttons()判断组合按键 判断move时候的左右键 结合 & 操作符
   10. 格式化字符串 QString( “ %1 %2 ” ).arg( 111 ).arg(222)
   11. 设置鼠标追踪 setMouseTracking(true);
3. 定时器1
   1. 利用事件 void timerEvent ( QTimerEvent \* ev)
   2. 启动定时器 startTimer( 1000) 毫秒单位
   3. timerEvent 的返回值是定时器的唯一标示 可以和ev->timerId 做比较
4. 定时器2
   1. 利用定时器类 QTimer
   2. 创建定时器对象 QTimer \* timer = new QTimer(this)
   3. 启动定时器 timer->start(毫秒)
   4. 每隔一定毫秒，发送信号 timeout ,进行监听
   5. 暂停 timer->stop
5. event事件
   1. 用途：用于事件的分发
   2. 也可以做拦截操作，不建议
   3. bool event( QEvent \* e);
   4. 返回值 如果是true 代表用户处理这个事件，不向下分发了
   5. e->type() == 鼠标按下 …
6. 事件过滤器
   1. 在程序将时间分发到事件分发器前，可以利用过滤器做拦截
   2. 步骤
      1. 1、给控件安装事件过滤器
      2. 2、重写 eventFilter函数 （obj ， ev）
7. QPainter 绘图
   1. 绘图事件 void paintEvent()
   2. 声明一个画家对象 QPainter painter(this) this指定绘图设备
   3. 画线、画圆、画矩形、画文字
   4. 设置画笔 QPen 设置画笔宽度 、风格
   5. 设置画刷 QBrush 设置画刷 风格
8. QPainter高级设置
   1. 抗锯齿 效率低
      1. painter.setRenderHint(QPainter::Antialiasing);
   2. 对画家进行移动
      1. painter.translate(100,0);
      2. 保存状态 save
      3. 还原状态 restore
   3. 如果想手动调用绘图事件 利用update
   4. 利用画家画图片 painter.drawPixmap( x，y，QPixmap( 路飞) )
9. QPaintDevice绘图设备
   1. QPixmap QImage QBitmap(黑白色) QPicture QWidget
   2. QPixmap 对不同平台做了显示的优化
      1. QPixmap pix( 300,300)
      2. pix.fill( 填充颜色 )
      3. 利用画家 往pix上画画 QPainter painter( & pix)
      4. 保存 pix.save( “路径”)
   3. Qimage 可以对像素进行访问
      1. 使用和QPixmap差不多 QImage img(300,300,**QImage::Format\_RGB32**);
      2. 其他流程和QPixmap一样
      3. 可以对像素进行修改 img.setPixel(i,j,value);
   4. QPicture 记录和重现 绘图指令
      1. QPicture pic
      2. painter.begin(&pic);
      3. 保存 pic.save( 任意后缀名 )
      4. 重现 利用画家可以重现painter.drawPicture(0,0,pic);
10. QFile 对文件进行读写操作
    1. QFile进行读写操作
    2. QFile file( path 文件路径)
    3. 读
       1. file.open(打开方式) QIODevice::readOnly
       2. 全部读取 file.readAll() 按行读 file.readLine() atend()判断是否读到文件尾
       3. 默认支持编码格式 utf-8
       4. 利用编码格式类 指定格式 QTextCodeC
       5. QTextCodec \* codec = QTextCodec::codecForName("gbk");
       6. //ui->textEdit->setText( codec->toUnicode(array) );
       7. 文件对象关闭 close
    4. 写
       1. file.open( QIODevice::writeOnly / Append)
       2. file.write(内容)
       3. file.close 关闭
11. QFileInfo 读取文件信息
    1. QFileInfo info(路径)
    2. qDebug() << "大小：" << info.size() << " 后缀名：" << **info.suffix()** << " 文件名称："<<info.fileName() << " 文件路径："<< info.filePath();
    3. qDebug() << "创建日期：" << info.created().toString("yyyy/MM/dd hh:mm:ss");
    4. qDebug() << "最后修改日期："<<info.lastModified().toString("yyyy-MM-dd hh:mm:ss");