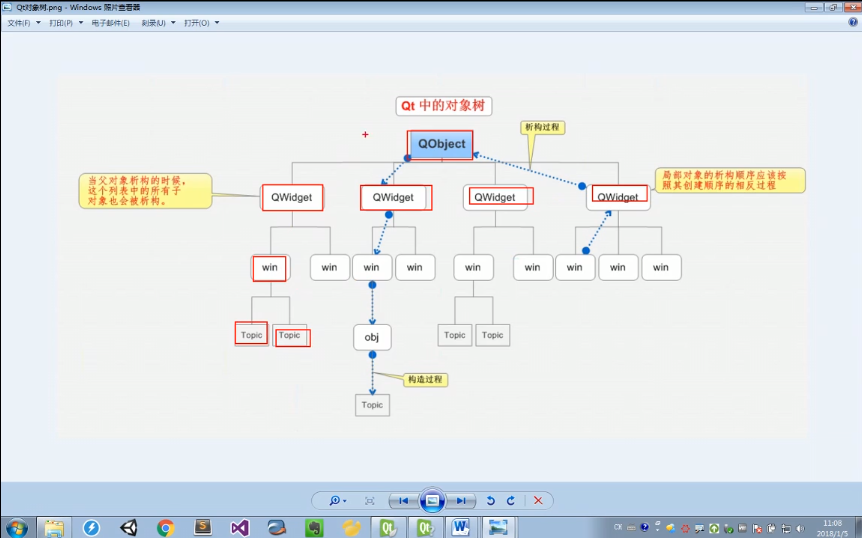
1. 命名规范
   1. 类名：首字母大写，单词和单词之间首字母大写
   2. 函数名，变量名称：首字母小写，单词和单词之间首字母大写
2. 快捷键
   1. 注释：选中后ctrl+/
   2. 运行：ctrl+r
   3. 编译：ctrl+b
   4. 字体缩放：ctrl+鼠标滚轮
   5. 查找：ctrl+f
   6. 整行移动：ctrl+shift+↑或者↓
   7. 帮助文档：fn+F1
   8. 自动对齐：ctrl+i
   9. 同名之间的.h和.cpp切换：fn+F4
3. 创建一个Qt程序
   1. 点击创建项目后，选择项目路径以及给项目起名称
   2. 名称：不能有中文字符，不能有空格
   3. 项目路径：不能有中文字符
   4. 默认创建有窗口类：起一个名称
   5. 基类有三种选择
      1. QWidget：一个最基础的类，是下面两个的父类
      2. QMainWindow：窗口类
      3. QDialog：对话框
   6. main函数
      1. QApplication a 应用程序对象，有且仅有一个
      2. 窗口类名称 w 实例化一个窗口对象
      3. w.show（） 调用show函数，显示窗口
      4. return a.exect（） 让应用程序对象进入消息循环机制中，使代码阻塞到当前行
4. 按钮控件常用API
   1. 创建 :QPushButton \*button=new QPushButton;
   2. 另一种创建方法QPushButton \*button=new QPushButton（“文字”，this）；
   3. 设置父亲 setParent();
   4. 设置文本 setText(“文字”)
   5. 设置位置 move（宽，高）；
   6. 重新制定窗口大小 resize（宽，高）
   7. 设置窗口标题 setWindowTitle（“文字”）
   8. 固定窗口大小 setFixedSize （宽，高）
   9. 设置按钮大小 button->resize（宽，高）
5. 对象模型（对象树）

在Qt中创建对象的时候会提供一个Parent对象指针，下面来解释这个parent到底是干什么的

* 1. QObject是以对象树的形式组织起来的

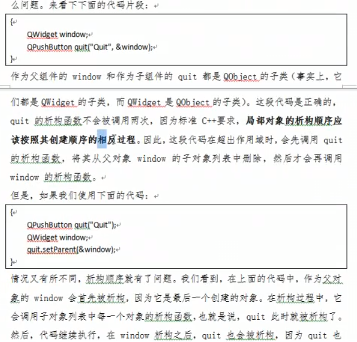
当你创建一个QObject对象时，会看到QObject的构造函数接收一个QObject指针作为参数，这个参数就是parent，也就是父对象指针。这相当于，**在创建QObject对象时，可以提供一个其父对象，我们创建的这个QObject对象会自动添加到其父对象的children（）列表。其父对象析构时，这个列表中所有的对象都会被析构。（注意，这里的父对象并不是继承意义上的父类！）**

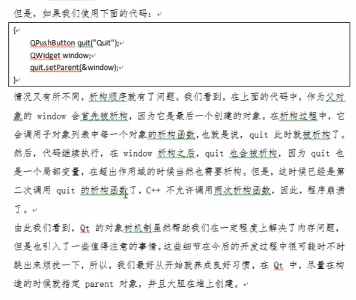
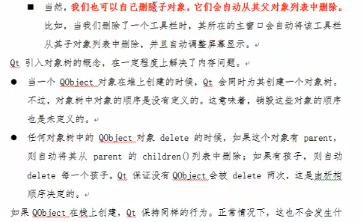
这种机制在GUI程序设计中相当有用。例如，一个按钮有一个QShortcut（快捷键）对象作为其子对象，当我们删除按钮时，这个快捷键理应一并被删除，这是合理的。



* 1. QWidget是能够在屏幕上显示的一切组件的父类

**QWidget继承自QObject，因此也继承了这种对象树关系。一个孩子自动的成为父组件的一个子组件。因此，他会显示在父组件的坐标系统中，被父组件的边界剪裁。**例如，当用户关闭一个对话框时，应用程序将其删除，那么，我们希望属于这个对话框的按钮、图标等应该一起被删除。事实就是如此，因为这些都是对话框的子组件。

* 1. 当创建的对象在堆区时，如果指定的父亲是QObject派生下来的类或者QObject子类派生下来的类，引以不用管理释放的操作，将对象会放入对象树中。一定程度上简化了内存回收机制
  2. 



1. Qt窗口坐标系

以左上角为原点（0，0），X向右增加，Y向下增加

对于嵌套窗口，其坐标是相对于父窗口来说的

控件的坐标看其左上角

1. 信号（signal）和槽（slot）
   1. 链接的函数connec
   2. 参数
      1. 参数1：信号的发送者
      2. 参数2：发送的信号（函数的地址：&类：：函数）
      3. 参数3：信号的接收者
      4. 参数4：处理的槽函数（函数的地址）
   3. 松散耦合
2. 自定义信号和槽
   1. 自定义信号
      1. 写到signals下
      2. 返回void
      3. 需要声明，不需要实现
      4. 可以有参数，可以重载
   2. 自定义槽函数
      1. 写到public slot下或者public或者全局函数
      2. 返回void
      3. 需要声明，也需要实现
      4. 可以有参数，可以重载
   3. 触发自定的信号

emit 对象->信号

* 1. 案例：老师下课后，触发饿了信号，学生响应信号，请客吃饭
  2. 当自定义的信号和槽出现重载
     1. 需要利用函数指针，明确指向函数的地址
     2. void 类名：：\*signal（信号或槽的特征）=&类名：：信号或槽
  3. 信号可以连接信号
  4. 断开信号disconnect（）
  5. 拓展
     1. 信号可以连接信号
     2. 一个信号可以连接多个槽函数
     3. 多个信号可以连接一个槽函数
     4. 信号和槽函数参数类型必须一一对应
     5. 信号和槽的参数的个数，可以不一致
     6. 信号的参数需要多于槽函数，且至少包含槽函数的参数类型
     7. 信号与信号连接，前一个的信号参数个数和类型需要多于或等于后一个
  6. Qt4版本写法
     1. connect（信号的发送者，SIGNAL（信号），信号的接收者，SLOT（槽函数））
     2. 优点：参数直观
     3. 缺点：参数类型不做检测，bug不容易发现

1. QString转成char\*
   1. 在QString的对象后面添加.toUtf8（）将其转为QByteArray
   2. 再添加.data（）转为char\*
2. QMainWindow
   1. 菜单栏（最多只有一个）
      1. QMenuBar \*bar=menuBar（）；
      2. setMenuBar（bar）；
      3. QMenu \*fileMenu=bar->addMenu（“文件”）；创建菜单
      4. QAction \*newAction=fileMenu->addAction（“新建”）创建菜单项
      5. fileMenu->addSeparator（）；
   2. 工具栏（可以有多个）
      1. QToolBar \*toolbar=new QToolBar（this）；
      2. addToolBar（默认停靠区域，toolbar）；Qt：：lefttoolbararea
      3. 设置只允许停靠范围setAllowedAreas（）；
      4. 设置浮动setFloatable（true/false）；
      5. 设置移动setMovable（true/false）;
      6. 添加菜单项 toolbar->addAction(newmenu)；
      7. 添加控件toolbar->addWidgt（）;
   3. 状态栏（至多一个）
      1. QStatusBar\*stBar= statusBar();
      2. 设置到窗口中 setStatusBar(stBar);
      3. 放置标签控件（左）

QLabel \*lable=new QLabel("提示信息",this);

stBar->addWidget(lable);

* + 1. 放置标签控件（右）

QLabel \*lable2=new QLabel("右侧提示信息",this);

stBar->addPermanentWidget(lable2);

* 1. 铆接部件 浮动窗口（可以有多个）
     1. QDockWidget
     2. addDockWidget（默认停靠区域，浮动窗口指针）
     3. 设置后期停靠区域setAllowedAreas（）；
  2. 设置核心部件（只能有一个）
     1. setCentralWidget（edit）；

1. 资源文件
   1. 将图片文件 拷贝到项目位置下
   2. 创建一个新文件

右键项目

添加新文件

Qt->Qt recourse File

给文件资源起名

Open in editor 编辑资源

添加前缀，添加文件

* 1. 使用

控件名->setIcon(QIcon（“地址“）)；

1. 对话框
   1. 分类
      1. 模态对话框 （不可以对其他窗口进行操作）
         1. 创建 QDialog \*dlg=new QDialog（this）；

dlg->exec();

* + 1. 非模态对话框
       1. 创建QDialog \*dlg=new QDialog (this);

dlg->show();

* + - 1. 释放dlg->setAttribute（Qt：：WA\_DeleteOnClose）
    1. 消息对话框
       1. QMessageBox静态成员函数 创建对话框
       2. 错误（critical），信息（information），提问（question），警告（warning）
       3. 参数1：父亲，参数2：标题，参数3：显示内容，参数4：按键类型，参数5：默认关联回车按键
       4. 返回值也是StandardButton类型，利用返回值判断用户的输入
    2. 颜色对话框
       1. 构建 QColorDialog：：getColor（）；
       2. 返回值 QColor；
    3. 文件对话框
       1. 构建 QFileDialog：：getOpenFileName（父亲，标题，默认路径，过滤条件（“\*.文件类型”））
       2. 返回值 QFile
    4. 字体对话框
       1. 构建 QFontDialog：：getFont（bool值，QFont（））
       2. QFont.family：字体（QString）

QFont.pointSize：字号（int）

QFont.blod()：是否加粗（bool）

QFont.italic()：是否倾斜（bool）

1. 界面布局
   1. 实现登录窗口
      1. 利用布局方式，对窗口进行美化
      2. 选取wediget 进行布局：水平，垂直，栅格
      3. 给用户名、密码、登录、退出按钮进行布局
      4. 默认窗口和控件之间有9间隙，可以利用layoutLeftMargin调整
      5. 弹簧可以居中，也可以自定义弹簧的大小
      6. 每一个wediget都有一个水平和垂直的策略，可以调整大小
      7. 可以通过调整最大和最小窗口大小来实现窗口大小的锁定
   2. 控件
      1. 按钮
         1. QPushButton
         2. QToolButton 工具按钮，用于显示图片，如果想显示文字：toolButtonStyle突起的效果：autoRaise
         3. radioButton 单选按钮，设置默认 ui->radioButton->setChecked（true）;
         4. checkBox 多选按钮，监听状态，stateChange：0未选，1半选，2选中
      2. QListWidget 列表容器
         1. QListWidgetItem 一行内容
         2. ui->listwidget->addItem();放入内容
         3. 设置居中方式item->setTextAlignment（int）；
         4. QSringList 一组内容
         5. 可以利用addItems放入一组内容
      3. QTreeWidget 树控件
         1. 设置头 setHeaderLabels；
         2. 创建节点 QTreeWidgetItem
         3. 添加根节点addTopLevelItem
         4. 添加子节点 addChild
      4. QTableWidget 表格控件
         1. 设置列数 setColumnNumber（）；
         2. 设置水平表头 setHorizontalHeaderLabels（）；
         3. 设置行数 setRowCount（）；
         4. 设置正文setItem（行数，列数，new QTableWidget（内容））；
      5. 其他控件
         1. QstackedWidget栈控件

setCurrentIndex（）设置位置

* + - 1. comboBox下拉框

addItem（）；

* + - 1. Qlabel标签
         1. 显示图片setPixmap（）
         2. 显示gif setMovie（）；

Movie->start（）；

1. 封装自定义控件
   1. 添加新文件 Qt-设计师界面类
   2. .ui中加入控件
   3. 在Widget中使用控件
      1. 添加一个widget
      2. 提升为.ui文件
   4. 在新加入的文件中实现功能
2. Qt中的事件
   1. 鼠标事件
      1. 鼠标进入事件 enterEvent（QEvent \*ev）；
      2. 鼠标离开事件 leaveEvent (QEvent \*ev);
      3. 鼠标按下事件 mousePressEvent (QEvent \*ev);
      4. 鼠标松开事件 mouseRealseEvent (QEvent \*ev);
      5. 鼠标移动事件 mouseMoveEvent (QEvent \*ev);
      6. X坐标与Y坐标

ev->x(),ev->y()

* + 1. ev->button() 可以判断所有按键 Qt::LeftButton,Qt::RightButton
    2. ev->buttons判断组合按键，判断move时候的左右键，结合&操作符
    3. 格式化字符串QString （“%1，%2，）.arg(“”);
  1. 定时器
     1. 利用事件 void timerEvent（QTimerEvent \*ev）；
     2. 启动定时器 startTimer （int）毫秒单位
     3. timerEvnet的返回值是计时器的唯一
     4. 定时器类QTimer
     5. 创建定时器对象 QTimer \*timer=new QTimer；
     6. 启动定时器 timer->start(毫秒);
     7. 暂停timer->stop（）；
     8. 每隔一定毫秒，发送信号timeout，进行监听
  2. event事件
     1. 用途：用于事件的分发
     2. 也可以做拦截操作，但不建议
     3. 返回值是bool类型，boo event（QEvent \*ev）；
     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. ev中有一个type类型，可以知道发生了什么
  3. 事件过滤器
     1. 在程序将事件分发到事件分发器前，乐意利用过滤器拦截
     2. 步骤
        1. 给控件安装事件过滤器

//给label\_1安装事件过滤器ui->label\_1->installEventFilter(this);

* + - 1. 重写eventFilter函数bool eventFilter（QObject \*，QEvent\*）；

1. QPainter绘图
   1. 绘图事件 void paintEvent（）；
   2. 声明一个画家 QPainter painter（this） this指定画图设备
   3. 画东西 painter.draw
   4. 设置画笔 QPen
   5. 设置填充色 QBrush
   6. 高级设置
      1. 抗锯齿，但是效率低painter.setRenderHint（）；
      2. 对画家进行移动painter.translate（）；
      3. 对画家状态进行保存painter.save();
      4. 对画家状态进行还原painter.restore（）；
   7. 利用画家画图片 painter.drawPixmap（x，y，QPixmap（“图片地址”））；
   8. QPaintDevice绘图设备
      1. QPixmap 对不同的平台进行了显示的优化
         1. QPixmap pix（长，宽）
         2. pix.fill（填充背景色）
         3. QPainter painter（& pix）利用画家王pix上画画
         4. pix.save（路径）
      2. QBitmap 黑白色
      3. QImage可以对像素进行访问
         1. QImage img(长，宽,QImage::Format\_ARGB32);
         2. 其他流程与Pixmap一样
         3. 可以对像素进行设置

QRgb value=qRgb(255,0,0); img.setPixel(i,j,value);

* + 1. QPicture 记录和重现绘图指令
       1. QPictuer pic；
       2. painter.begin（pic）；向pic上画画
       3. painter.end（）；结束画画
       4. pic.save（）保存
       5. painter.drawPicture（x位置，y位置，pic）；重现操作

1. QFile对文件进行读写操作
   1. QFile file（“文件路径”）
   2. file.open（打开方式）
      1. QIODevice::ReadOnly只读模式
      2. QIODevice::Append用追加方式写
      3. QIODevice::WriteOnly全部重写
   3. 读取方式
      1. file.readAll（）全部读取
      2. file.readLine（）按行读
      3. file.atEnd（）判断是否读到尾
      4. 默认支持编码格式utf-8

利用编码格式类 指定格式 QTextCodec

QTextCodec \*codec=QTextCodec::codecForName("docx");

ui->textEdit->setText((codec->toUnicode(array)));

* 1. file.write（“内容”）；

1. QFileInfo 读取文件信息
   1. QFileInfo info（“文件路径”）
   2. 大小.size（）
   3. 后缀名.suffix()
   4. 文件名称.fileName()
   5. 文件路径.filePath()
   6. 创建日期.created()
   7. 最后修改日期lastModified()
   8. QDataTime可以用toString（）进行格式化