



Qt部件

徐枫

清华大学软件学院

feng-xu@tsinghua.edu.cn



课程主要内容

- 用户界面部件介绍
- 部件的布局管理
- 通用部件
- Qt Designer



用户界面部件介绍

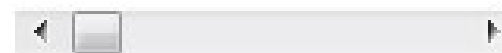


用户界面部件介绍

- 用户界面由一个个部件（widget）构成

Label

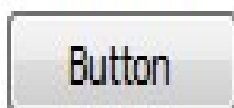
QLabel



QScrollBar

Line Editor

QLineEdit



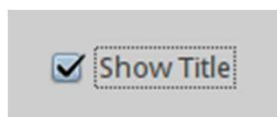
QPushButton



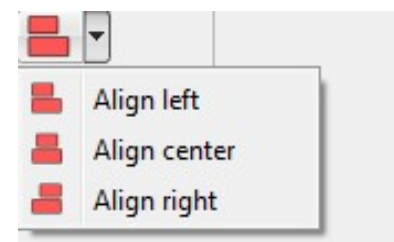
QDoubleSpinBox



QRadioButton



QCheckBox

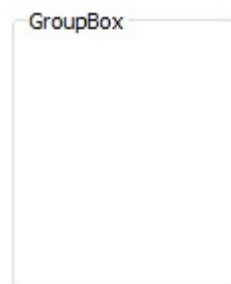


QToolButton

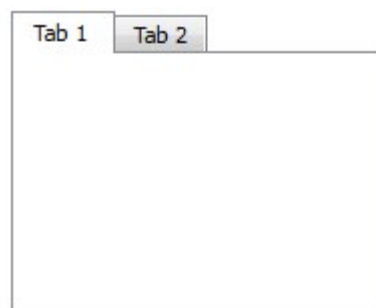


部件中的部件

- 部件被分层次放置



QGroupBox



QTabWidget

- 容器类提供可视化结构，同时也是具有一定功能的
 - 如 **QRadioButton**，需要用彼此间实现互斥，可以将多个QRadioButton放到一个GroupBox中



部件的特点

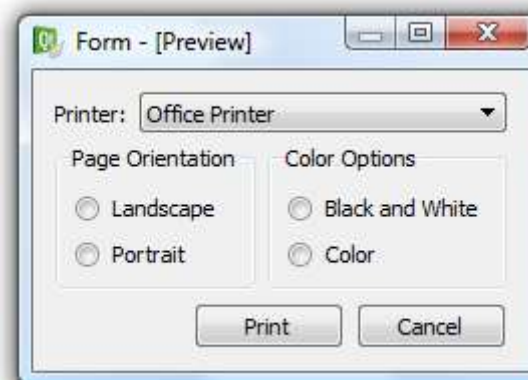
- 占据屏幕中一个矩形区域
- 从输入设备接收事件
 - 文本修改，项目被选中，按钮被点击
- 当部件产生变化时，发射信号
- 一个部件中可以包含其他部件
- 多个部件以层次式的方法组合构建



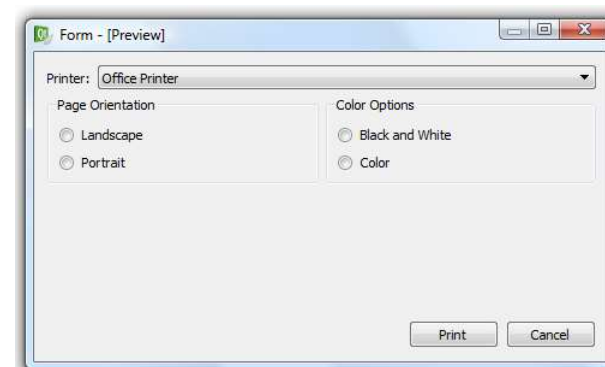
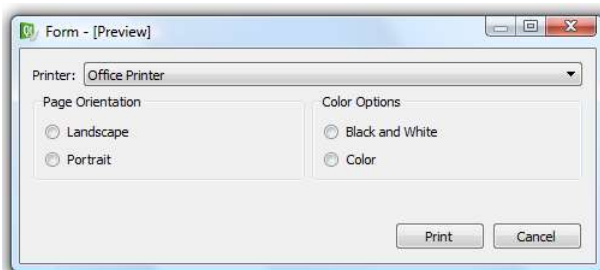
部件的布局管理

部件的布局管理

- 一个对话框例子



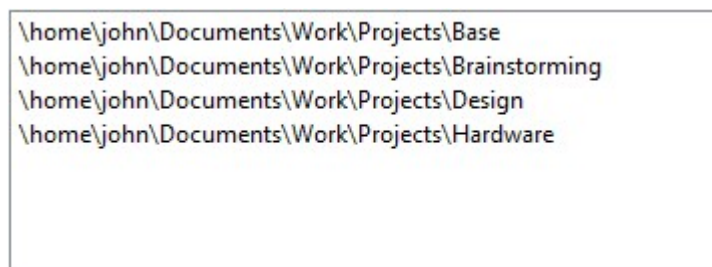
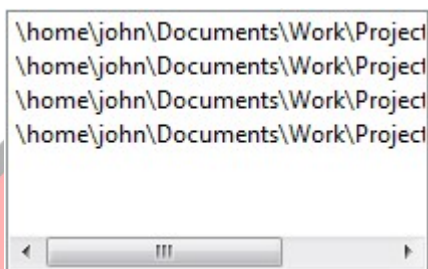
- 部件被放置在布局管理器（QLayout）中—使用户界面具有弹性易伸缩





弹性好在哪里？

- 让部件的大小适应内容



- 让部件适应翻译变化



Nyheter

- 让部件适应用户设置，
如字体设置等

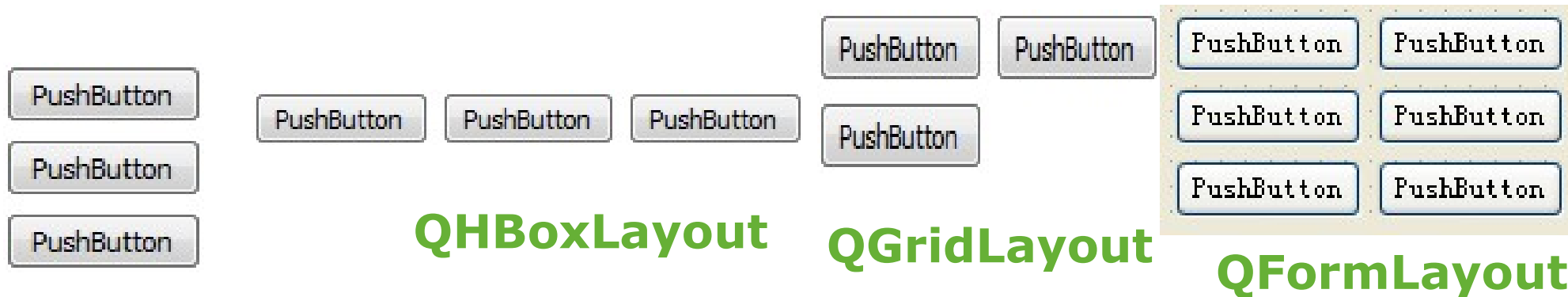




News



布局管理

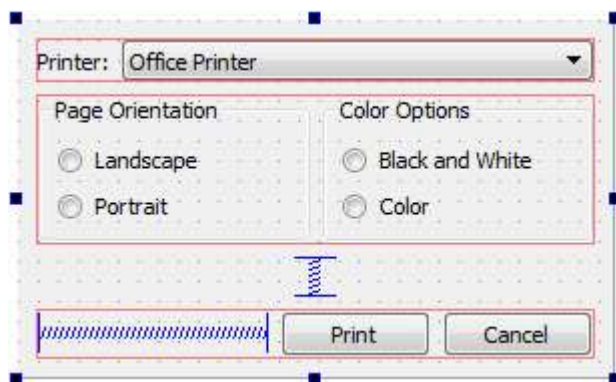
- 几种可用的布局



- 布局管理器和部件“协商”各个部件大小与位置
- 弹簧可以用来填充空白处  

一个对话框例子

- 对话框由多个层次的布局管理器和部件组成



注意：布局管理器并不是其管理的部件的父对象，但可以是其他布局管理器的父对象

Object	Class
Form	QWidget
horizontalLayout	QHBoxLayout
label	QLabel
printerBox	QComboBox
horizontalLayout_2	QHBoxLayout
cancelButton	QPushButton
horizontalSpacer	Spacer
printButton	QPushButton
horizontalLayout_3	QHBoxLayout
groupBox	QGroupBox
landscapeButton	QRadioButton
portraitButton	QRadioButton
groupBox_2	QGroupBox
bwButton	QRadioButton
colorButton	QRadioButton
verticalSpacer	Spacer

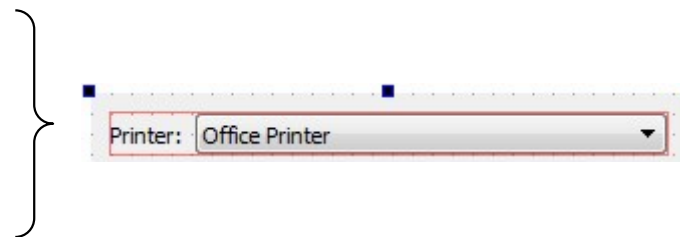
- 1、一般功能组件的父对象只能是一般容器类组件，而不能是布局管理器
- 2、布局管理器的父对象可以是一般容器类的组件或是布局管理器



对话框例子

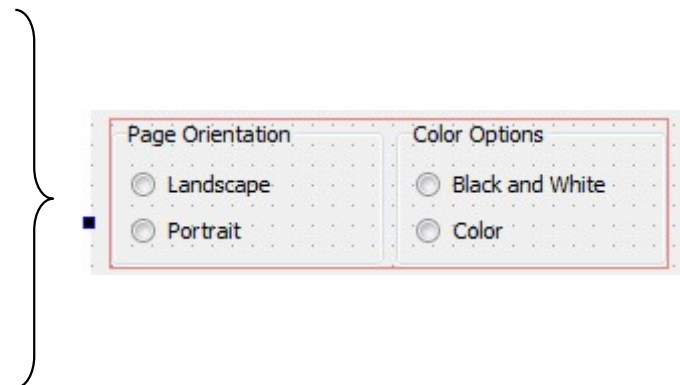
```
QVBoxLayout *outerLayout = new QVBoxLayout(this);
```

```
QHBoxLayout *topLayout = new QHBoxLayout();  
topLayout->addWidget(new QLabel("Printer:"));  
topLayout->addWidget(c=new QComboBox());  
outerLayout->addLayout(topLayout);
```



```
QHBoxLayout *groupLayout = new QHBoxLayout();
```

... (后面着重讲)

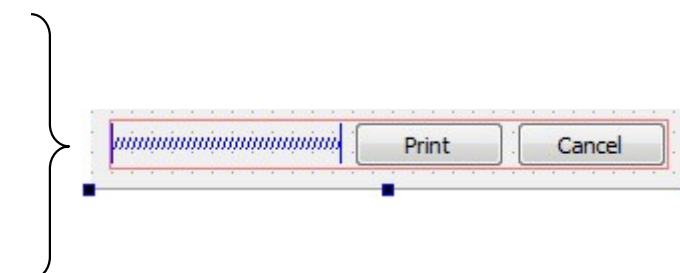


```
outerLayout->addLayout(groupLayout);
```

```
outerLayout->addSpacerItem(new QSpacerItem(...));
```

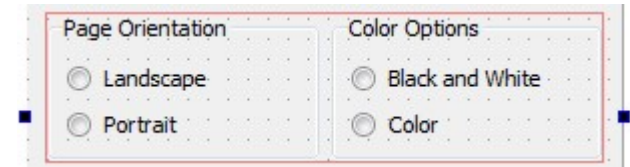


```
QHBoxLayout *buttonLayout = new QHBoxLayout();  
buttonLayout->addSpacerItem(new QSpacerItem(...));  
buttonLayout->addWidget(new QPushButton("Print"));  
buttonLayout->addWidget(new QPushButton("Cancel"));  
outerLayout->addLayout(buttonLayout);
```



对话框例子

- Horizontal box, 包含两个 group boxes, vertical boxes, radio buttons



```
QHBoxLayout *groupLayout = new QHBoxLayout();
```

```
QGroupBox *orientationGroup = new QGroupBox();
```

```
QVBoxLayout *orientationLayout = new QVBoxLayout(orientationGroup);
```

```
orientationLayout->addWidget(new QRadioButton("Landscape"));
```

```
orientationLayout->addWidget(new QRadioButton("Portrait"));
```

```
groupLayout->addWidget(orientationGroup);
```

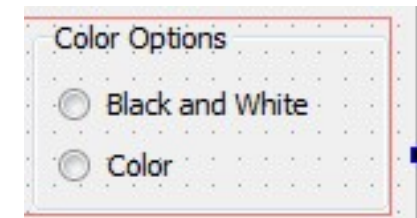
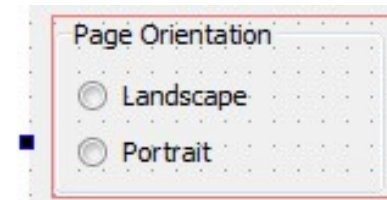
```
QGroupBox *colorGroup = new QGroupBox();
```

```
QVBoxLayout *colorLayout = new QVBoxLayout(colorGroup);
```

```
colorLayout->addWidget(new QRadioButton("Black and White"));
```

```
colorLayout->addWidget(new QRadioButton("Color"));
```

```
groupLayout->addWidget(colorGroup);
```





对话框例子

- 可以使用Qt设计器来建立同样的结构

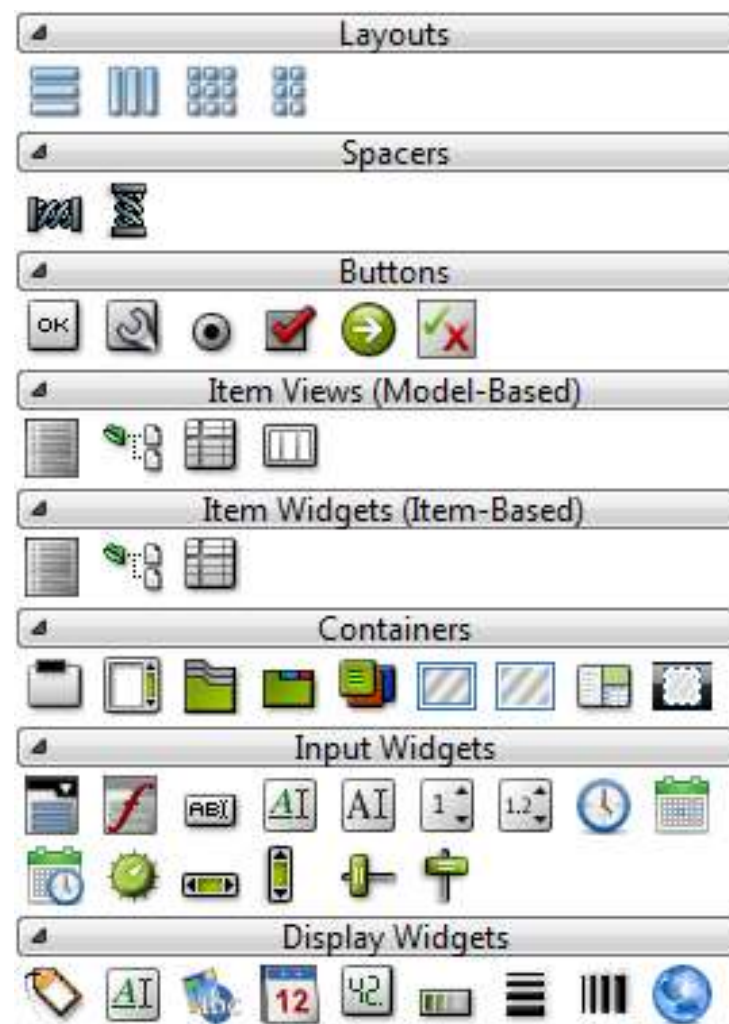
Object	Class
Form	QWidget
horizontalLayout	QHBoxLayout
label	QLabel
printerBox	QComboBox
horizontalLayout_2	QHBoxLayout
cancelButton	QPushButton
horizontalSpacer	Spacer
printButton	QPushButton
horizontalLayout_3	QHBoxLayout
groupBox	QGroupBox
landscapeButton	QRadioButton
portraitButton	QRadioButton
groupBox_2	QGroupBox
bwButton	QRadioButton
colorButton	QRadioButton
verticalSpacer	Spacer



通用部件

通用部件

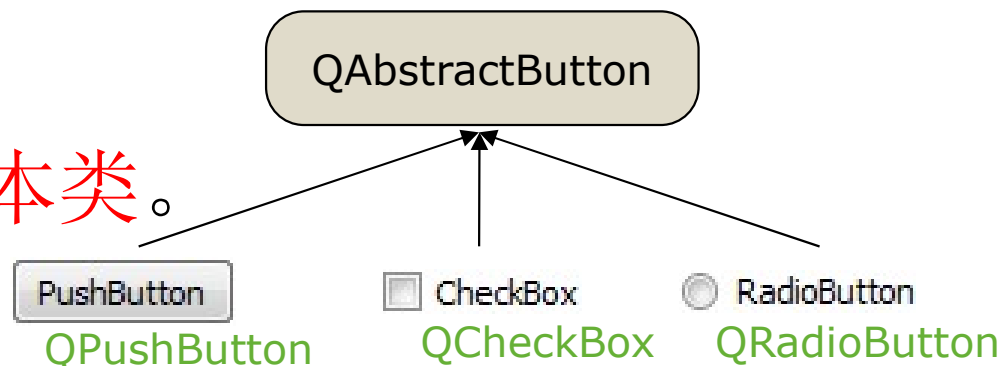
- Qt包含针对所有常见需求的大量通用部件
- Qt设计器中为部件组提供很好的概貌





通用部件—按钮

- 所有按钮继承自 **QAbstractButton** 这个基本类。



- 信号
 - **clicked()** – 当按钮被按下（并弹起后）发出。
 - **toggled(bool)** – 当按钮的状态发生改变时发出。
- 属性
 - **checkable** – 当按钮可检查时为真。使按钮激活。
 - **checked** – 当按钮被标记时为真。（用于复选或单选按钮）
 - **text** – 按钮的文本。
 - **icon** – 按钮的图标（可以和文本同时显示）。





通用部件—列表项部件

- **QListWidget**用于显示列表项
- 添加项目
 - addItem(QString) – 将项目附加到列表末端
 - insertItem(int row, QString) – 将项目插入到指定行
- 选择项目
 - selectedItems – 返回QListWidgetItem的选中的items, 使用 QListWidgetItem::text来形成文本
- 信号
 - itemSelectionChanged – 当选择状态改变时发出
- **QComboBox** 以更紧密的格式展示一个单选的项目列表。



QListWidget



QComboBox



通用部件—容器

- 容器部件用来结构化用户界面
- 一个简单的 **QWidget** 对象可当做容器来使用
- 设计器：将部件放置在容器中并为容器提供一个布局管理器
- 代码：为容器创建一个布局管理器并将部件添加进布局管理器（布局管理器以容器为父对象）

```
QGroupBox *box = new QGroupBox();  
QVBoxLayout *layout = new QVBoxLayout(box);  
layout->addWidget(...);  
...
```

GroupBox

QGroupBox

Tab 1

Tab 2

QTabWidget

QFrame



通用部件—输入部件

- 使用QLineEdit 实现单行文本输入
- 信号
 - textChanged(QString) – 文本状态改变时发出
 - editingFinished() – 部件失去焦点时发出
 - returnPressed() – 回车键被按下时发出
- 属性
 - text – 部件的文本
 - maxLength – 限定输入的最大长度
 - readOnly – 设置为真时文本不可编辑（仍允许复制）

Hello World

QLineEdit



通用部件—输入部件

- 使用QTextEdit 和 QPlainTextEdit 实现多行文本输入

- 信号

- textChanged() - 文本状态改变时发出

- 属性

- plainText – 无定义格式文本
- html – HTML格式文本
- readOnly – 设置为真时文本不可编辑



QTextEdit

- QComboBox 通过editable属性使其可编辑

- Signals

- editTextChanged(QString) – 当文本被编辑时发出

- 属性

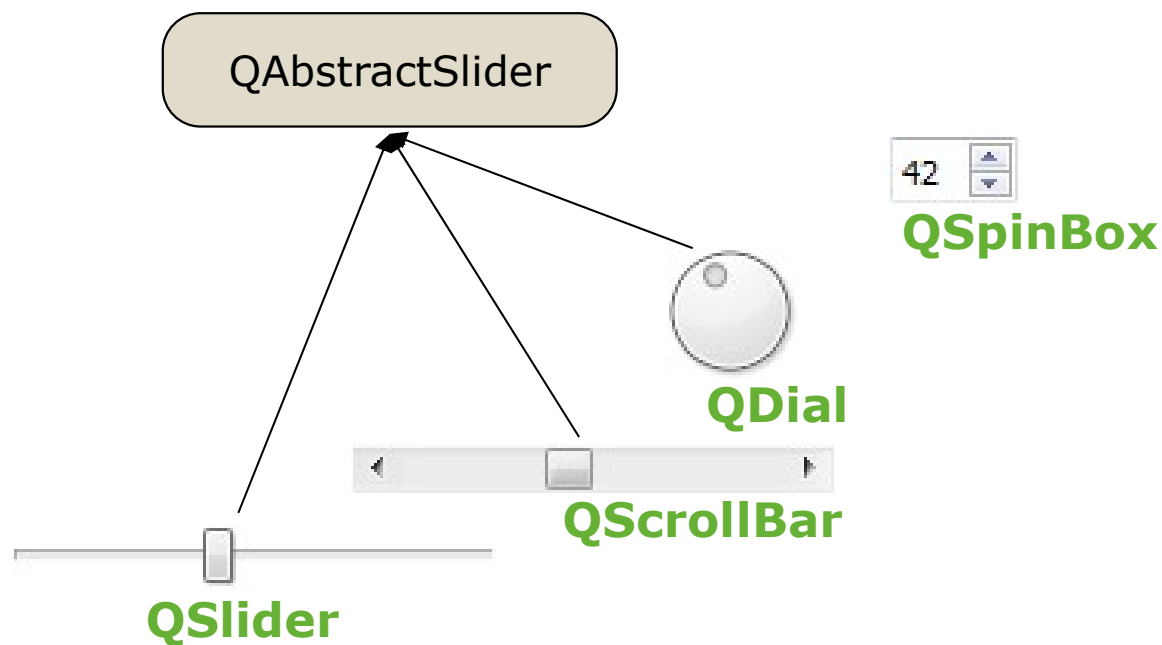
- currentText – combo box的当前文本



QComboBox

通用部件—输入部件

- 编辑整型数据有许多可选的输入部件
- 也有许多用于double, time和date类型的部件
 - 信号
 - valueChanged(int) – 当数值更新时发出
 - 属性
 - value – 当前值
 - maximum – 最大值
 - minimum – 最小值





通用部件—显示部件

- **QLabel** 部件显示文本或者图片

- 属性

- text – 标签文本
 - pixmap – 显示的图片

- 接口

- setText(), setPixmap()

HelloWorld
QLabel



QLabel

- **QLCDNumber** 用于显示整形数值

- 属性

- intValue – 显示的  (用display(int)函数进行设置)
- QLCDNumber



通用部件—属性

- 所有部件有一系列继承自QWidget类的共同属性

- enabled – 用户交互可用或不可用



- visible – 显示或不显示(show 或hide函数)



- 这些属性同时影响到子部件
例如使一个容器部件不可用时:





QMessageBox

- 信息框是可以显示提示信息，并接受用户按钮输入的一种对话框
- 信息框使用方式一：静态函数
 - `StandardButton QMessageBox::warning (QWidget * parent, const QString & title, const QString & text, StandardButtons buttons = Ok, StandardButton defaultButton = NoButton)`
 - Parent: 父部件指针
 - Title: 标题
 - Text: 提示文本
 - Buttons: 提示框中的按钮，可用或（|）运算添加多个按钮
 - defaultButton: 默认选中的按钮
 - 类似函数还有 `QMessageBox::information (...)`, `QMessageBox::critical(...)`, `QMessageBox::question(...)`, `QMessageBox::about(...)`, ...



QMessageBox

```
StandardButton ret = QMessageBox::warning(this, QObject::tr("My  
Application"), QObject::tr("The document has been modified.\n" "Do  
you want to save your changes?"), QMessageBox::Save |  
QMessageBox::Discard | QMessageBox::Cancel, QMessageBox::Save);
```

```
switch (ret) {  
    case QMessageBox::Save:    // Save was clicked  
        break;  
    case QMessageBox::Discard: // Don't Save was clicked  
        break;  
    case QMessageBox::Cancel:  // Cancel was clicked  
        break;  
    default:    // should never be reached  
        break;  
}
```



QMessageBox

- 信息框使用方式二：构造函数

- QMessageBox::QMessageBox (Icon icon, const QString & title, const QString & text, StandardButtons buttons = NoButton, QWidget * parent = nullptr, Qt::WindowFlags f = Qt::Dialog | Qt::MSWindowsFixedSizeDialogHint)
 - icon: 图标, 可取值为QMessageBox::NoIcon, QMessageBox::Information, QMessageBox::Question, QMessageBox::Warning, QMessageBox::Critical
 - Title: 标题
 - Text: 提示文本
 - Buttons: 提示框中的按钮, 可用或 (|) 运算添加多个按钮
 - parent: 父组件指针
 - F: 窗口系统属性



QMessageBox

```
QMessageBox message(QMessageBox::NoIcon, QObject::tr("My  
Application"), QObject::tr("The document has been modified.\n" "Do  
you want to save your changes?"), QMessageBox::Save |  
QMessageBox::Discard | QMessageBox::Cancel);
```

```
switch (message.exec()) {  
    case QMessageBox::Save:    // Save was clicked  
        break;  
    case QMessageBox::Discard: // Don't Save was clicked  
        break;  
    case QMessageBox::Cancel:  // Cancel was clicked  
        break;  
    default: // should never be reached  
        break;  
}
```



Qt Designer

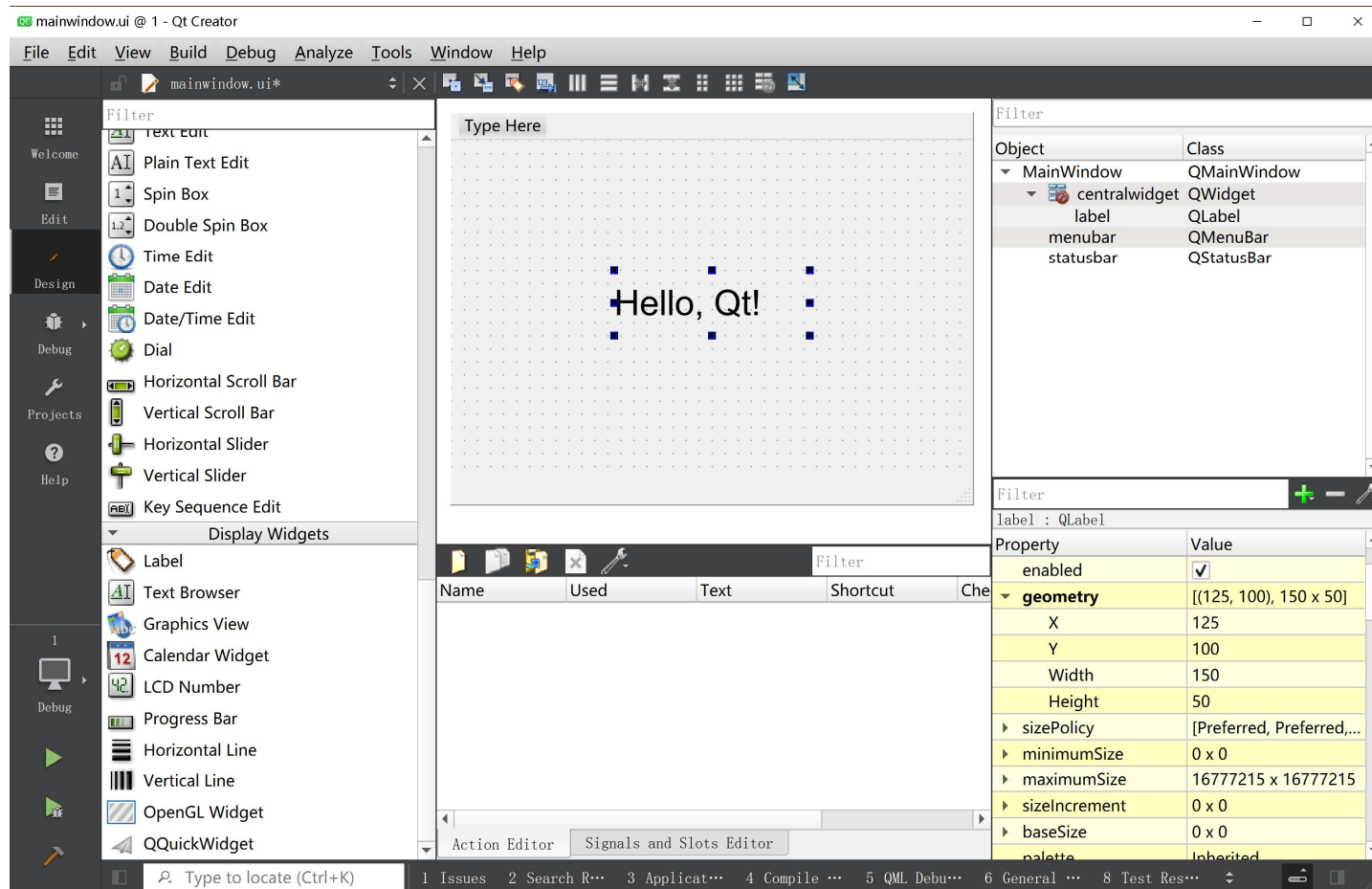


Qt Designer

- Qt应用程序除了使用手工编写代码的方式外，还可以用Qt Designer来完成
- Qt Designer曾是一个独立的Qt桌面工具，现在集成于Qt Creator中
- 只需要拖动相应的控件
- 输出为.ui文件，内容其实就是XML
- Uic编译器把.ui 文件转换成.h文件
 - myproject.ui -> ui_myproject.h

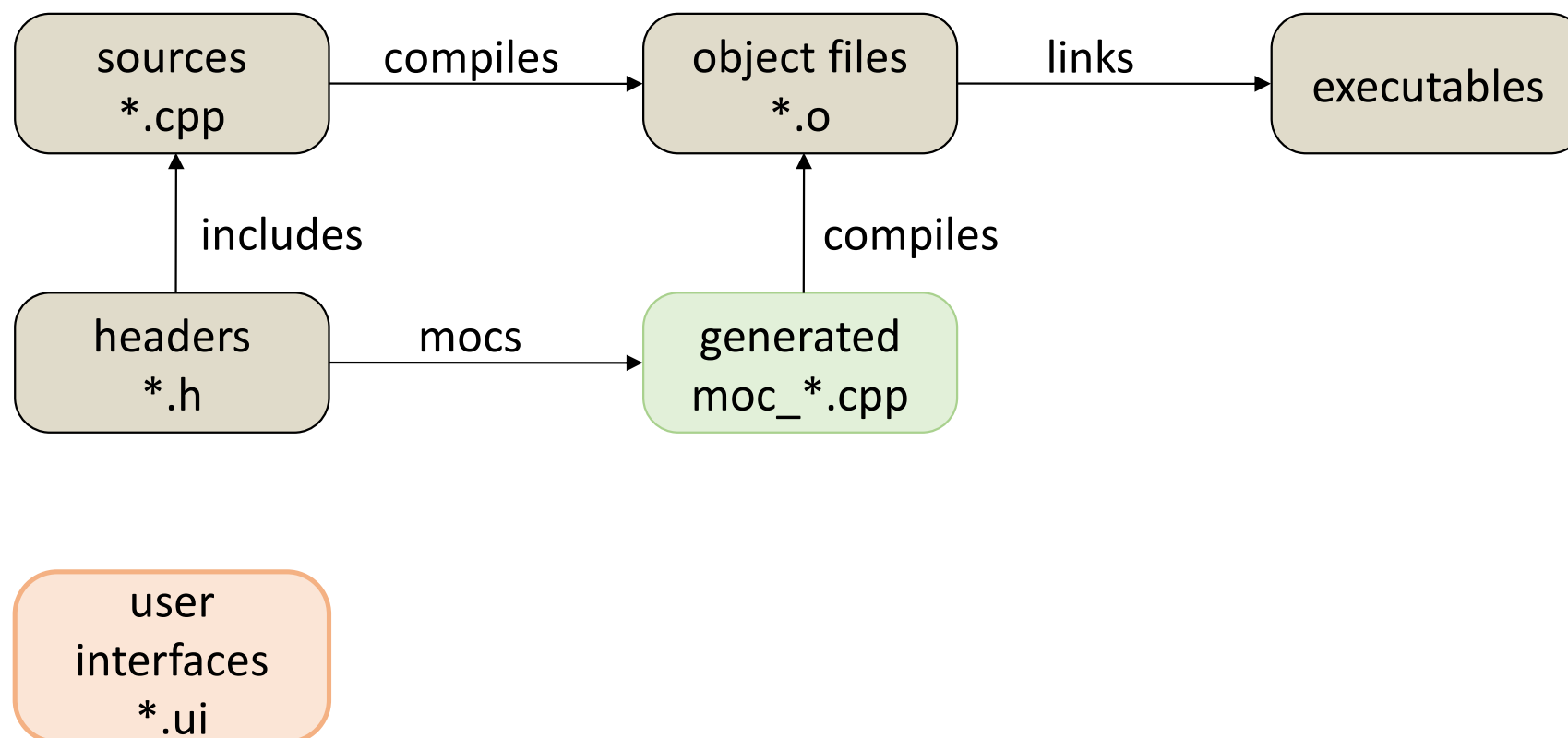
使用Qt Designer

- 双击目录树中以.ui结尾的界面文件，调出Qt Designer界面
- 从左侧的分组的组件面板中将**Label**组件拖放到设计窗体上，通过拖放设置大小，双击编辑文字内容，在右下角属性窗口内设置属性值。



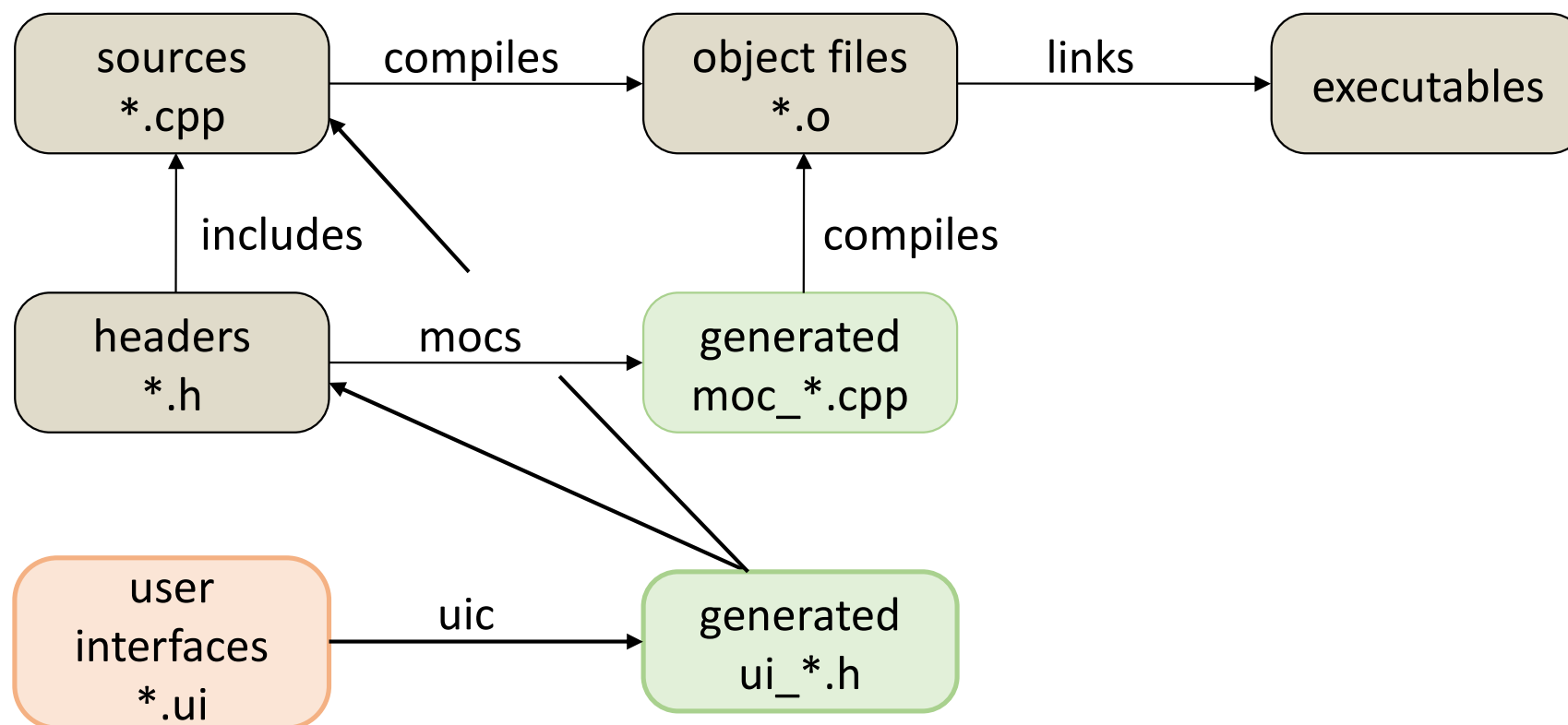


设计器介绍





设计器介绍





使用代码

包含uic编译器
生成的.h文件

```
#ifndef QTTEST_H
#define QTTEST_H

#include "ui_qttest.h"
#include <QtWidgets/QMainWindow>

class QtTest : public QMainWindow
{
    Q_OBJECT

public:
    QtTest(QWidget *parent = 0);
    ~QtTest();

private:
    Ui::QtTestClass ui;
};

#endif // QTTEST_H
```

基本上是一个标
准的 **QWidget** 派
生类

一个 **Ui::QtTestClass**
类对象**ui**，
指向所有部件



使用代码

调用函数 **setupUi**,
生成所有父窗体
(**this**)的子窗体部件

```
#include "qttest.h"

QtTest::QtTest(QWidget *parent)
: QMainWindow(parent)
{
    ui.setupUi(this);
}

QtTest::~QtTest()
{
}
```

实例化类
Ui:: QtTestClass 为 **ui**

删除 **ui**对象



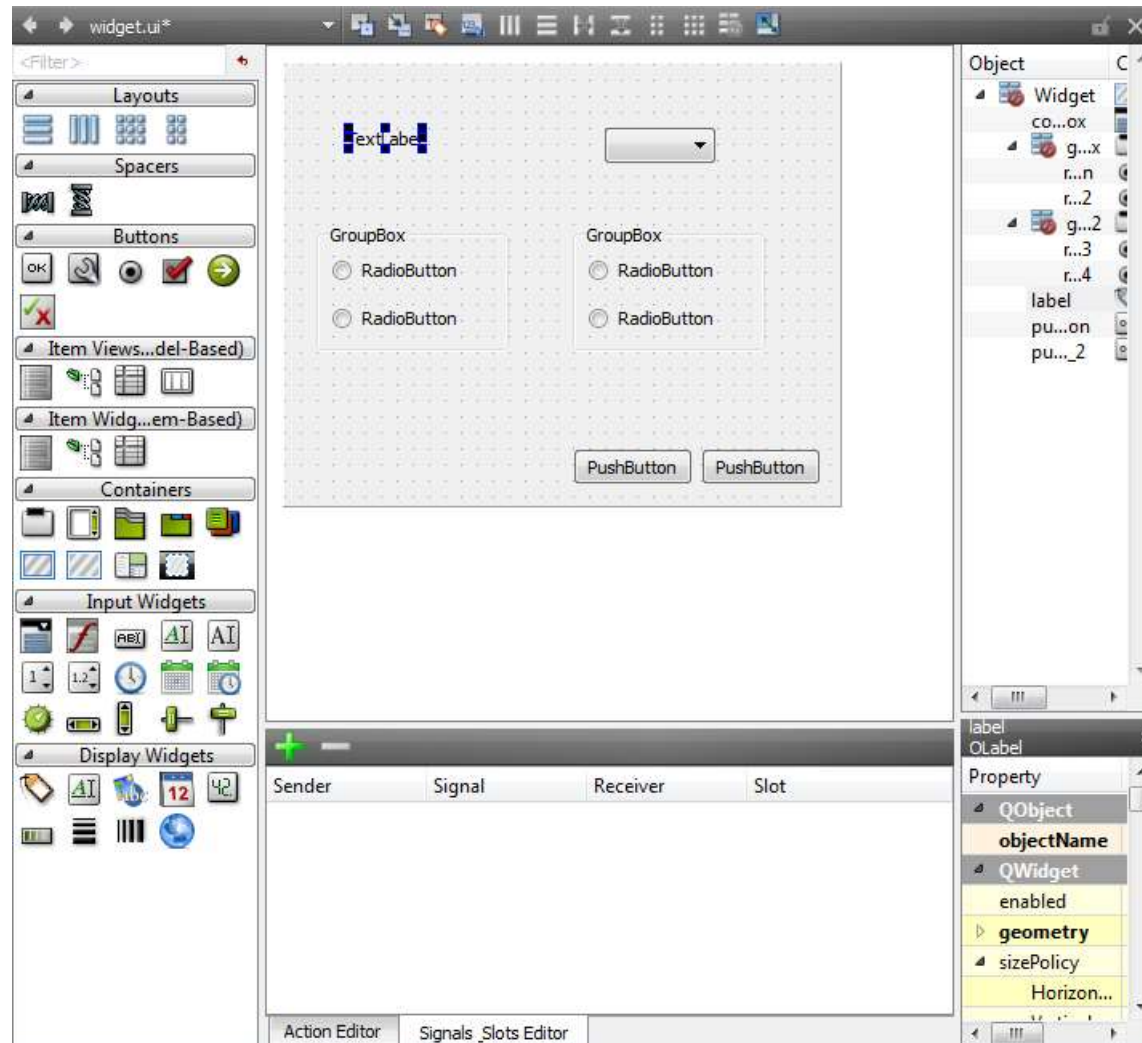
使用设计器

- 基本工作流程
 - 粗略地放置部件在窗体上
 - **从里到外**进行布局，添加必要的弹簧
 - 进行信号连接
 - 在代码中使用
- 在整个过程中不断修改编辑属性
- 实践创造完美!



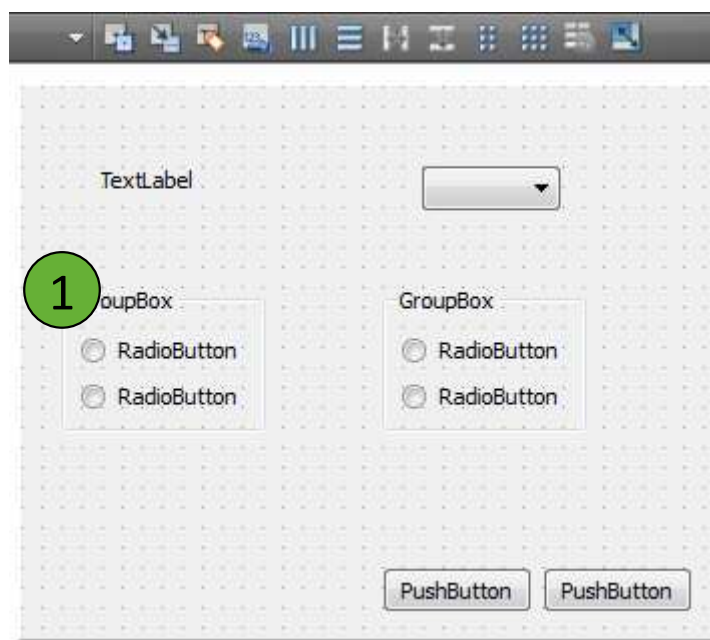
使用设计器

粗略地放置部件在窗体上



使用设计器

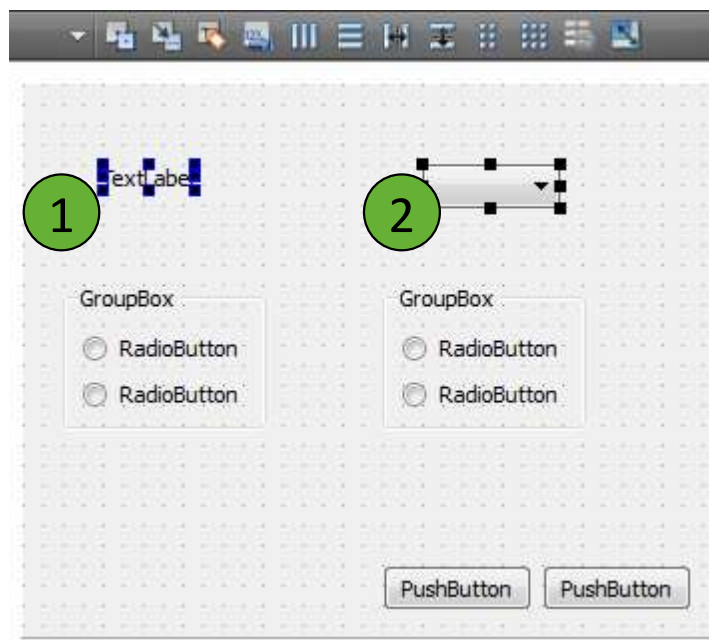
- 从里到外进行布局，添加必要的弹簧



1. 选中每一个 group box, 2. 应用垂直布局管理

使用设计器

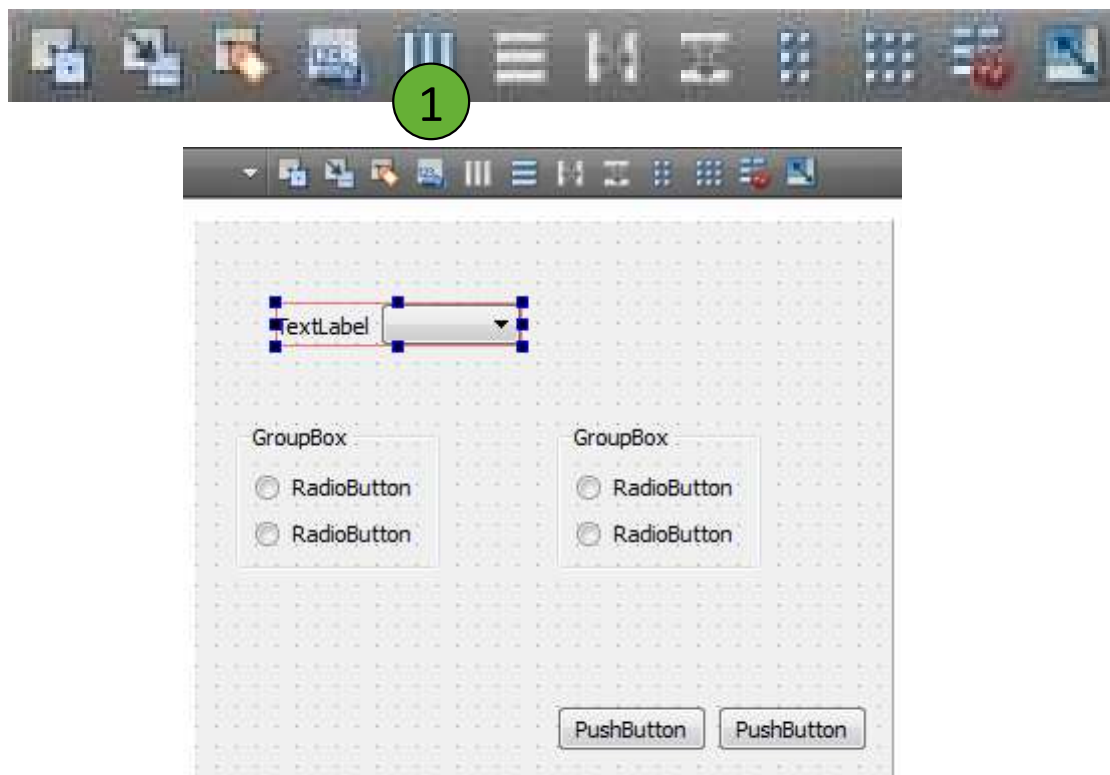
- 从里到外进行布局，添加必要的弹簧



1. 选中label (click), 2. 选中combobox (Ctrl+click)

使用设计器

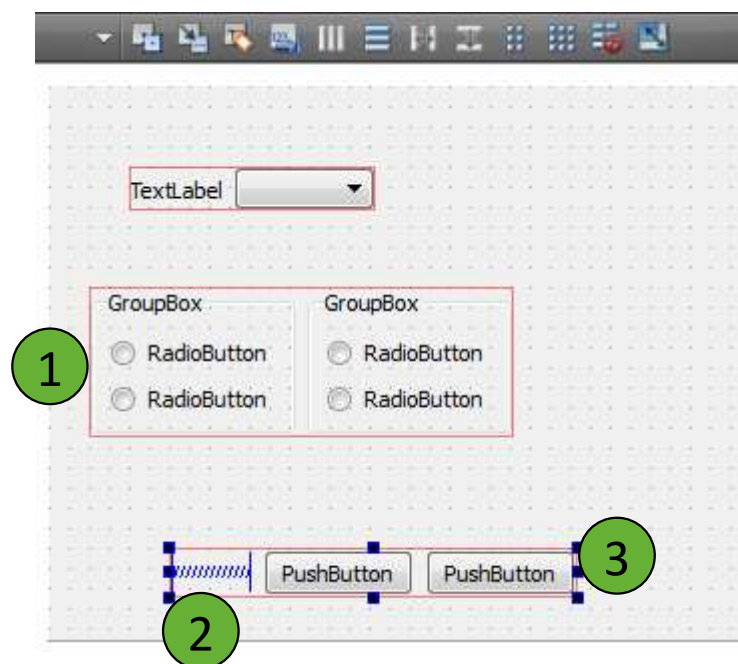
- 从里到外进行布局，添加必要的弹簧



1. 应用一个水平布局管理

使用设计器

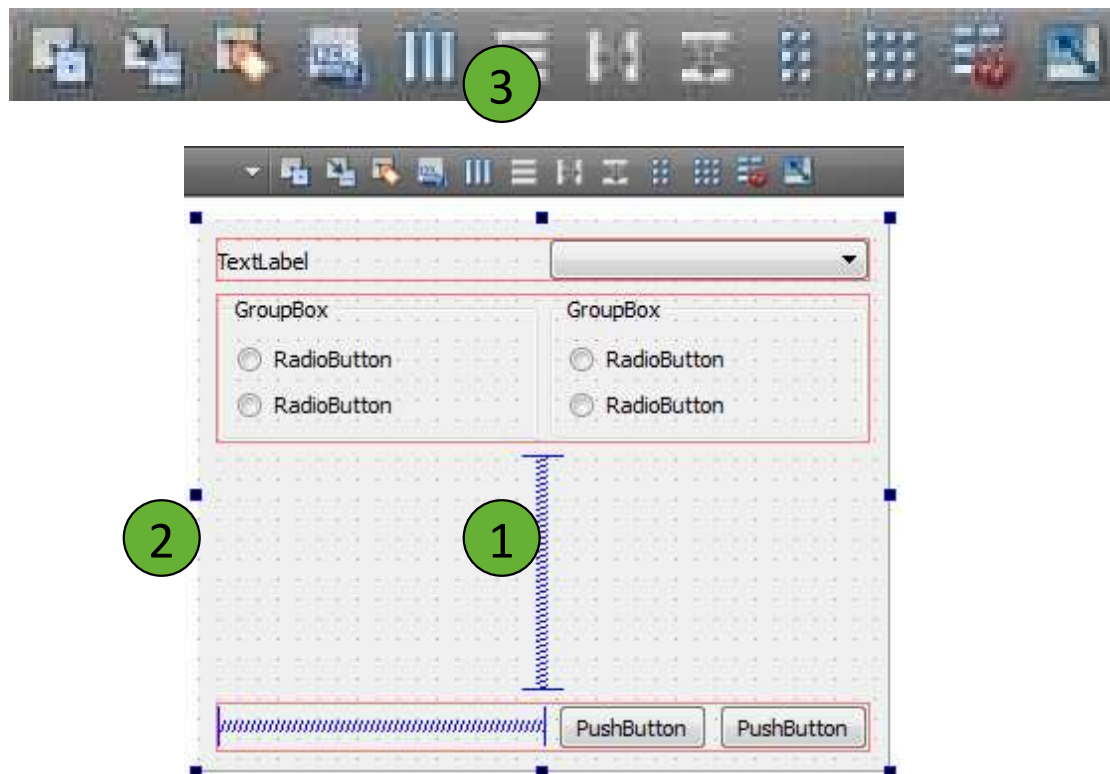
- 从里到外进行布局，添加必要的弹簧



1. 选中2个group box并应用一个水平布局管理,
2. 添加一个水平弹簧,
3. 将弹簧和按钮选中并应用一个水平布局管理

使用设计器

- 从里到外进行布局，添加必要的弹簧

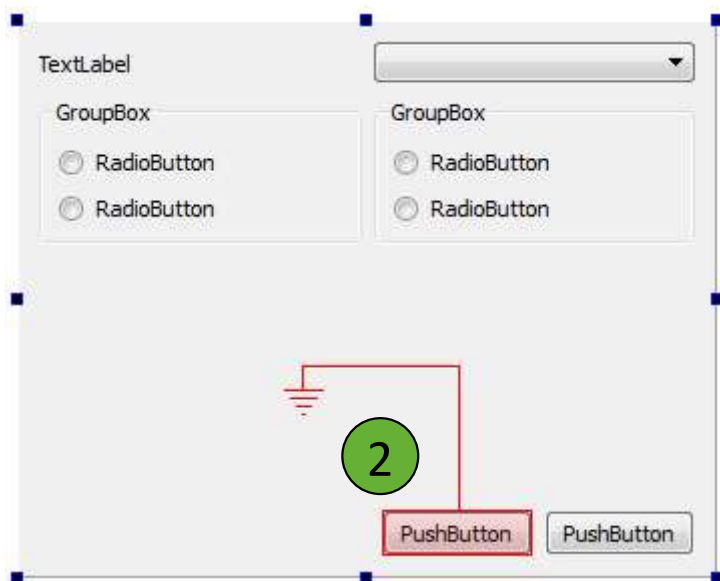


1. 添加一个垂直弹簧
2. 选中窗体本身
3. 应用一个垂直布局管理

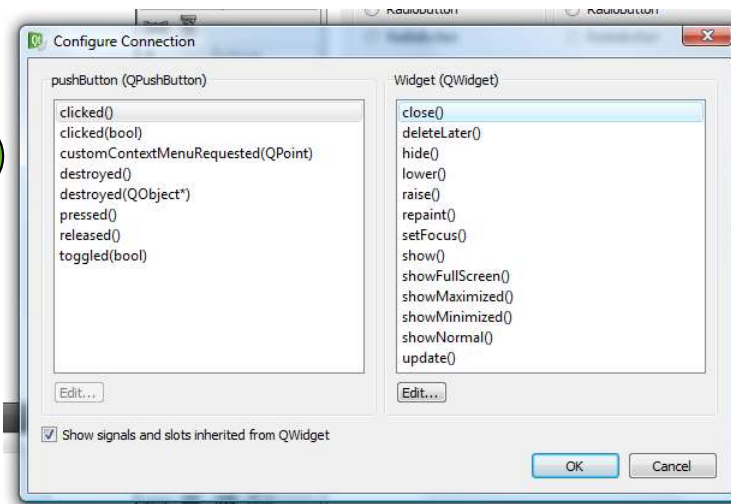
使用设计器

- 进行信号连接(部件之间)

1



3



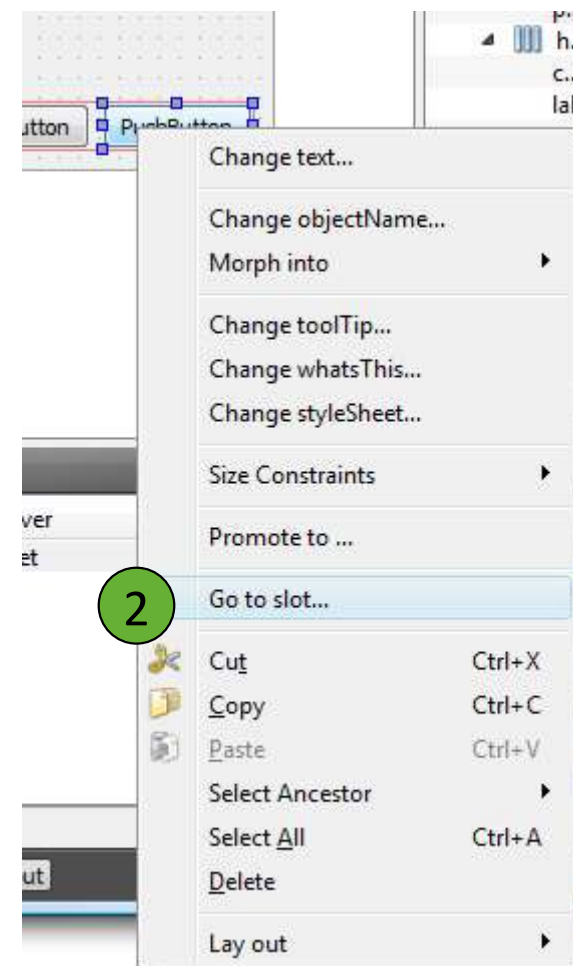
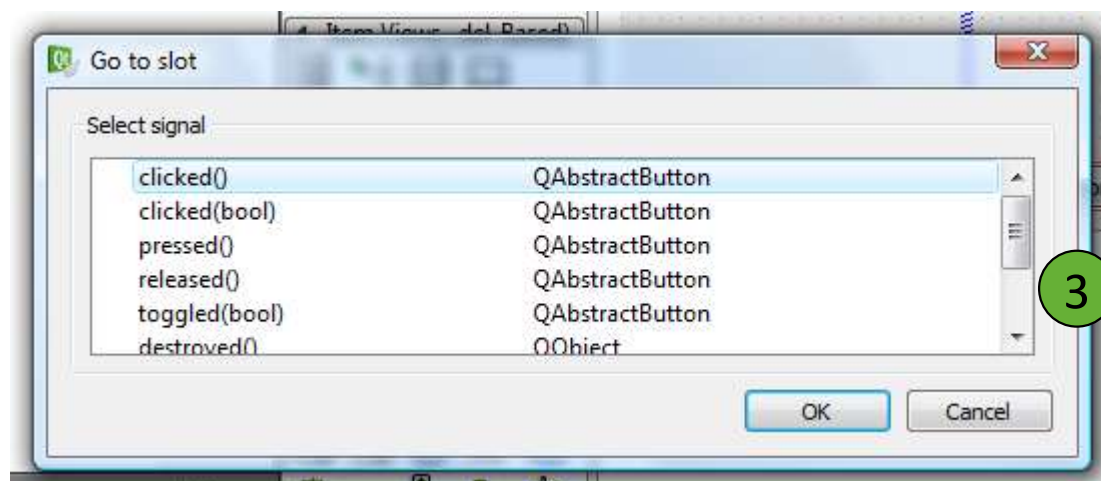
4



1. 转到signals and slot 编辑模式,
2. 从一个部件拖放鼠标到另一个部件,
3. 选中signal and slot, 4. 在信号和槽编辑器中查看结果

使用设计器

- 进行信号连接(到你的代码中)



1. 在widget editing 模式中
2. 右击一个部件并选择 Go to slot...
3. 选择一个信号来连接到你的代码



使用设计器

- 在代码中使用
- 通过ui类成员的使用，访问其所有子部件

```
class Widget : public QWidget {  
    ...  
private:  
    Ui::Widget *ui;  
};
```

```
void Widget::memberFunction()  
{  
    ui->pushButton->setText(...);  
}
```



谢谢！



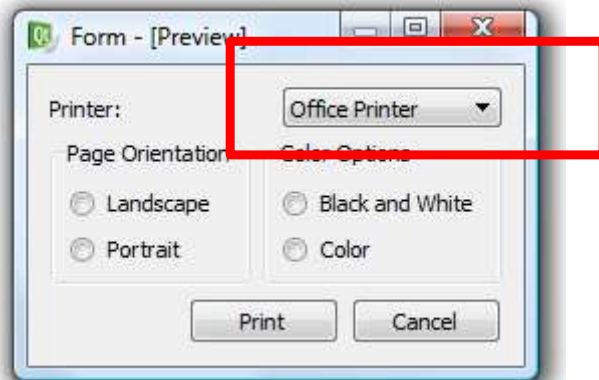
部件的尺寸策略



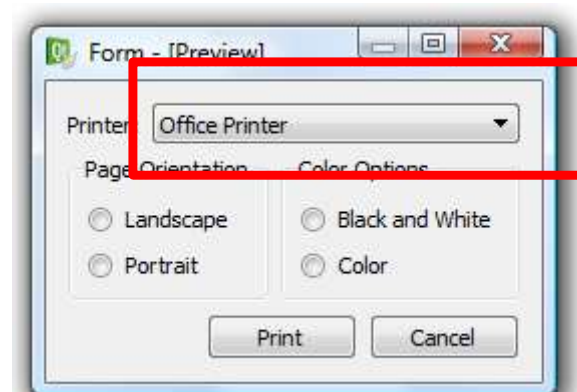
尺寸策略

- 布局是在**布局管理器**和部件间进行协调的过程
- 布局管理器提供布局结构
 - 水平布局和垂直布局
 - 网格布局
 - 表格布局
- 部件则提供
 - **各个方向上的尺寸策略**
 - **最大和最小尺寸**

尺寸的策略



Void setSizePolicy(QSizePolicy::Policy horizontal, QSizePolicy::Policy vertical)
printerList->setSizePolicy(QSizePolicy::Expanding, QSizePolicy::Fixed)





尺寸的策略

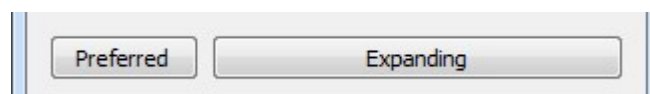
- 每一个部件都有一个尺寸大小的示意（hint），给出水平和垂直方向上的尺寸的策略
 - **Fixed** – 规定了widget的尺寸，固定大小（最严格）
 - **Minimum** – 规定了可能的最小值，可增长
 - **Maximum** – 规定可能的最大值，可缩小
 - **Preferred** – 给出最佳值，但不是必须的，可增长可缩小
 - **Expanding** – 同preferred，但希望增长
 - **MinimumExpanding** – 同minimum，但希望增长
 - **Ignored** – 忽略规定尺寸， widget得到尽量大的空间

如果?

- 2个 preferred 相邻



- 1个 preferred, 1个 expanding



- 2个 expanding 相邻



- 空间不足以放置widget (fixed)





关于尺寸的更多内容

- 可用最大和最小属性更好地控制所有部件的大小
- `maximumSize` –最大可能尺寸
- `minimumSize` –最小可能尺寸

```
ui->pushButton->setMinimumSize(100, 150);  
ui->pushButton->setMaximumHeight(250);
```