QT简介

姜桂飞 张圣林

南开大学软件学院

- Qt简介
- Qt的使用
- Qt深入理解
- Qt的应用

Qt简介

什么是Qt?

什么是Qt?

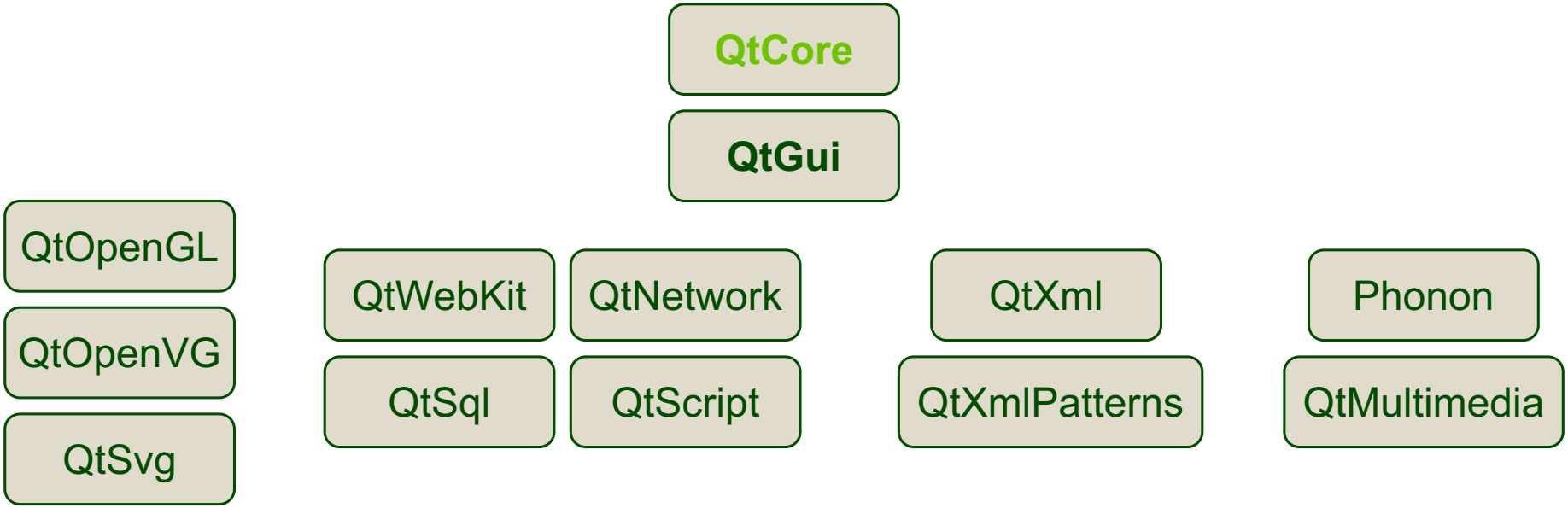
“Qt 是一个用C++编写的跨平台开发框架.”

原来用作用户界面开发，现可用作所有的开发

例如：Databases, XML, WebKit, multimedia, networking, OpenGL, scripting, non-GUI...

什么是Qt?

- Qt由模块构建



什么是QT?

- Qt用宏（macros）和内省（introspection）扩展了C++

```
foreach (int value, intList) { … }
```

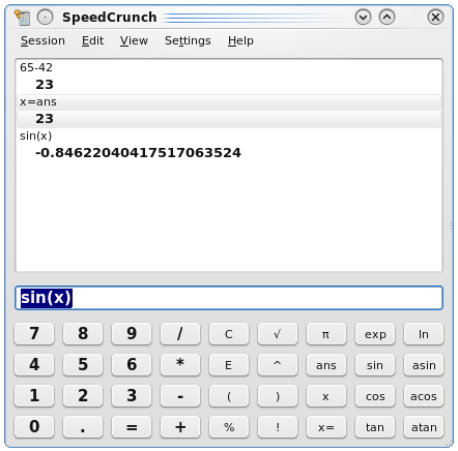
```
QObject *o = new QPustButton;
o->metaObject()->className(); // 返回 ”QPushButton”
```

```
connect(button, SIGNAL(clicked()), window, SLOT(close()));
```

- 所有的代码仍然是简明C + +

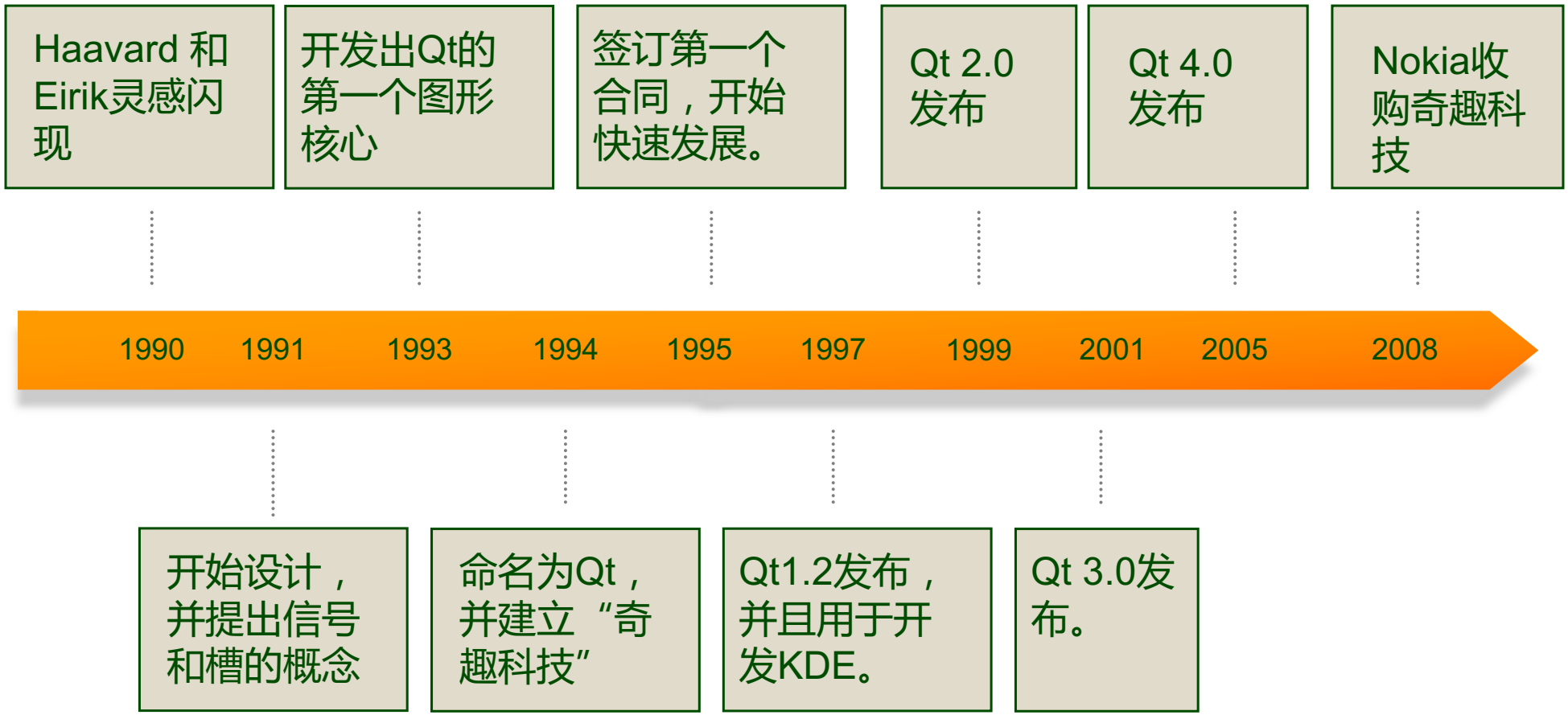
Qt的目的

- 一次编写，到处编译
- 根据不同平台的本地观感生成相应的本地应用



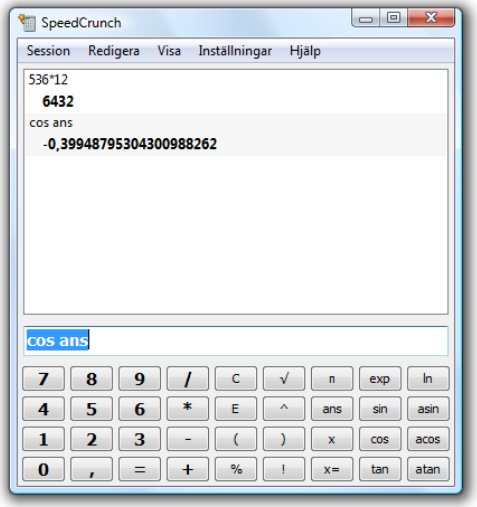
- 简单地使用API，高开发效率，开放性，使用有趣

Qt的历史

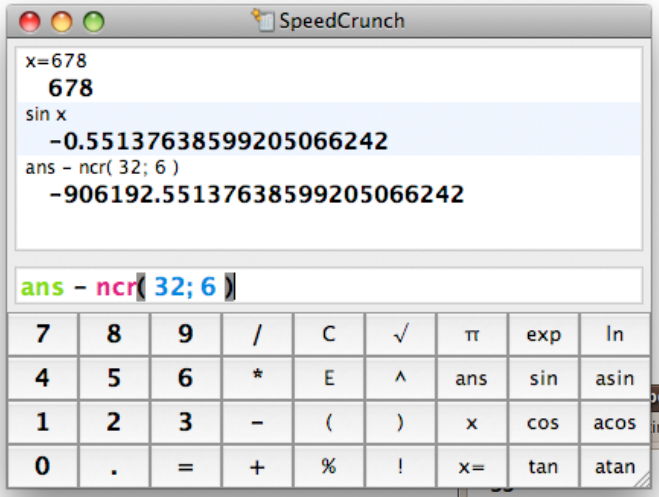

Haavard 和 Eirik灵感闪现
开发出Qt的第一个图形核心
签订第一个合同，开始快速发展。
Qt 2.0 发布
Qt 4.0 发布
Nokia收购奇趣科技
1990
1991
1993
1994
1995
1997
1999
2001
2005
2008
开始设计，并提出信号和槽的概念
命名为Qt，并建立“奇趣科技”
Qt1.2发布，并且用于开发KDE。
Qt 3.0发布。

Qt的跨平台—桌面平台

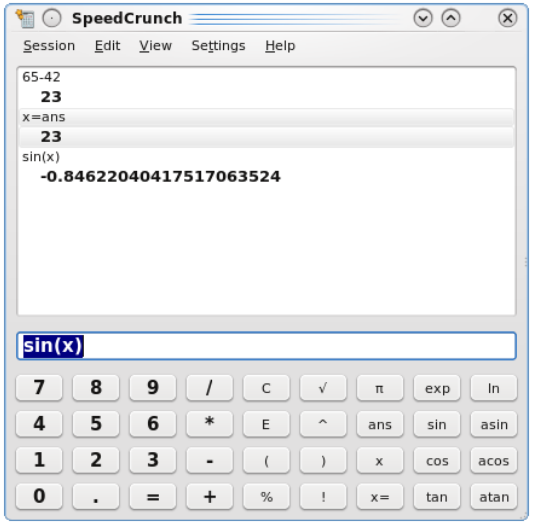
- Windows



- Mac OS X



- Linux/Unix X11

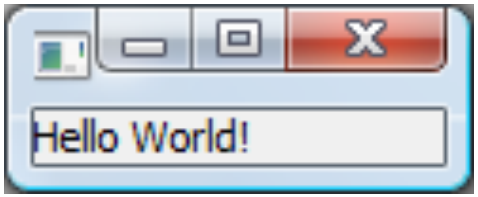


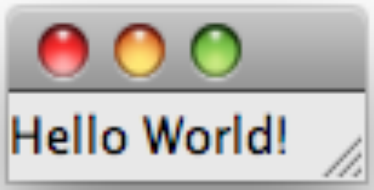
Qt的跨平台—嵌入式平台

- Windows CE
- Symbian
- Maemo
- 嵌入式Linux

- Qt简介
- Qt的使用
- Qt深入理解
- Qt的应用

Qt的代码结构--Hello World





Qt的代码结构--Hello World

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel l( "Hello World!" );
    l.show();
    return app.exec();
}
```

Qt的代码结构--Hello World

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel l( "Hello World!" );
    l.show();
    return app.exec();
}
```

Qt的代码结构--Hello World

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel l( "Hello World!" );
    l.show();
    return app.exec();
}
```

Qt的代码结构--Hello World

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel l( "Hello World!" );
    l.show();
    return app.exec();
}
```

Qt的代码结构--Hello World

```
#include <QApplication>
#include <QLabel>

int main( int argc, char **argv )
{
    QApplication app( argc, argv );
    QLabel l( "Hello World!" );
    l.show();
    return app.exec();
}
```

Qt开发工具集

1. Qt Creator
2. Qt Designer
3. Qt Linguist
4. Qt Assisant
5. Qt Demos

信号(Signal)与槽(Slot)初探

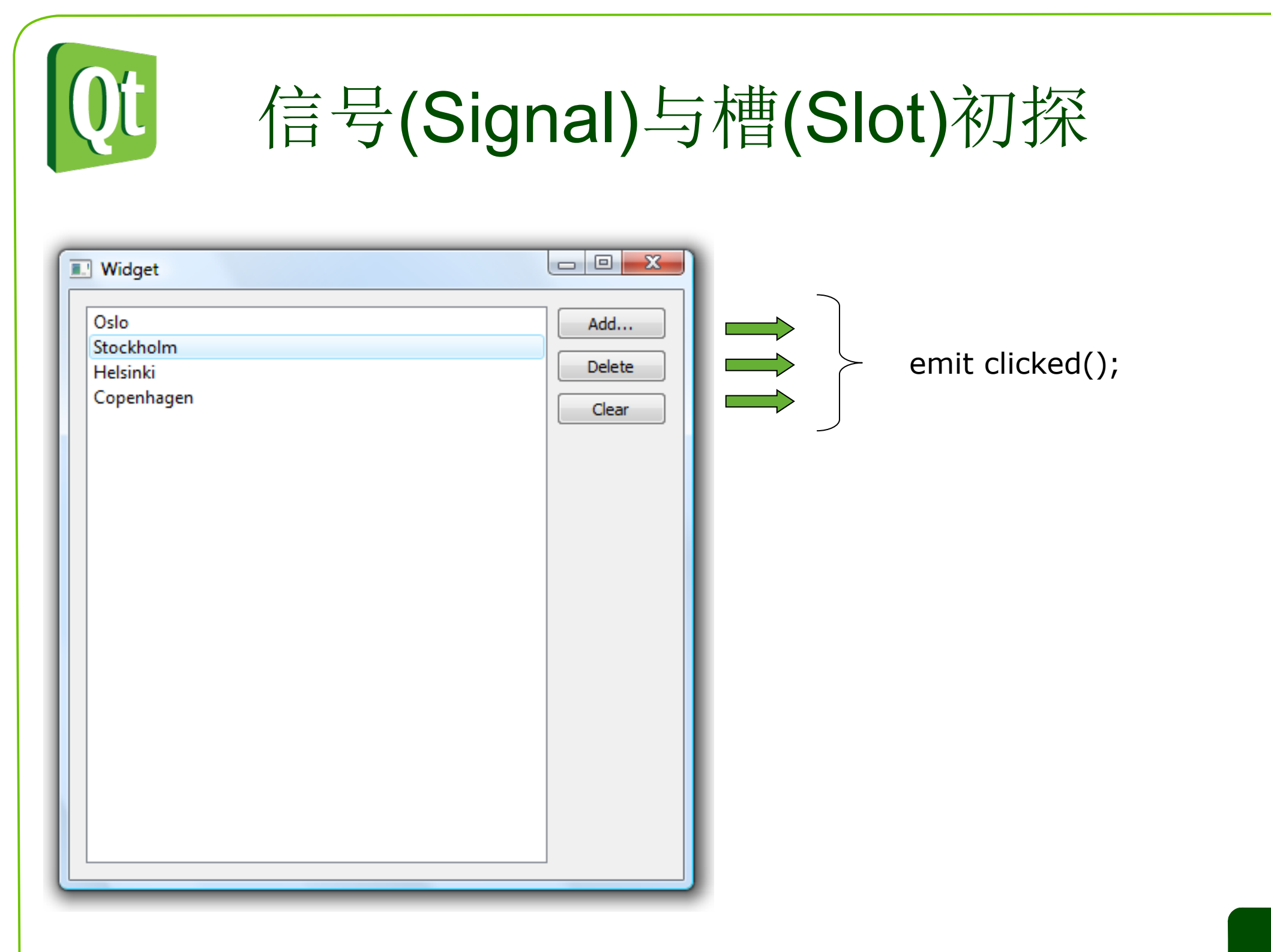
QT提供的一种在对象间进行通讯的技术。

动态地或松散地将事件和状态变化联系起来。

信号和槽机制是使Qt运作的元素。

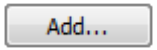
信号(Signal)与槽(Slot)初探

- ✓ 类似于windows中的消息和消息响应
- ✓都是通过C++类成员函数实现的
- ✓信号和槽是通过连接实现相互关联的
- ✓包含信号或槽的类必须从QObject继承



信号(Signal)与槽(Slot)初探

信号(Signal)与槽(Slot)初探



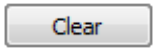

```
{
   ...
   emit clicked();
   ...
}
```

```
{
   QString newText =
      QInputDialog::getText(this,
                            "Enter text", "Text:");

   if( !newText.isEmpty() )
      ui->listWidget->addItem(newText);
}
```

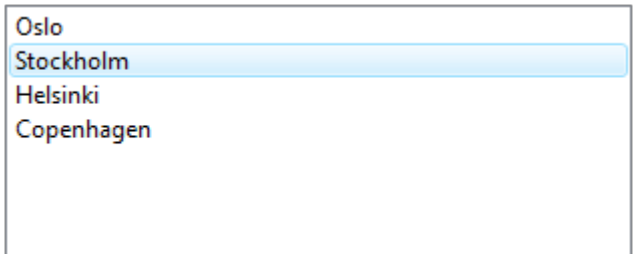

```
{
   ...
   emit clicked();
   ...
}
```

```
{
   foreach (QListWidgetItem *item,
         ui->listWidget->selectedItems())
   {
      delete item;
   }
}
```

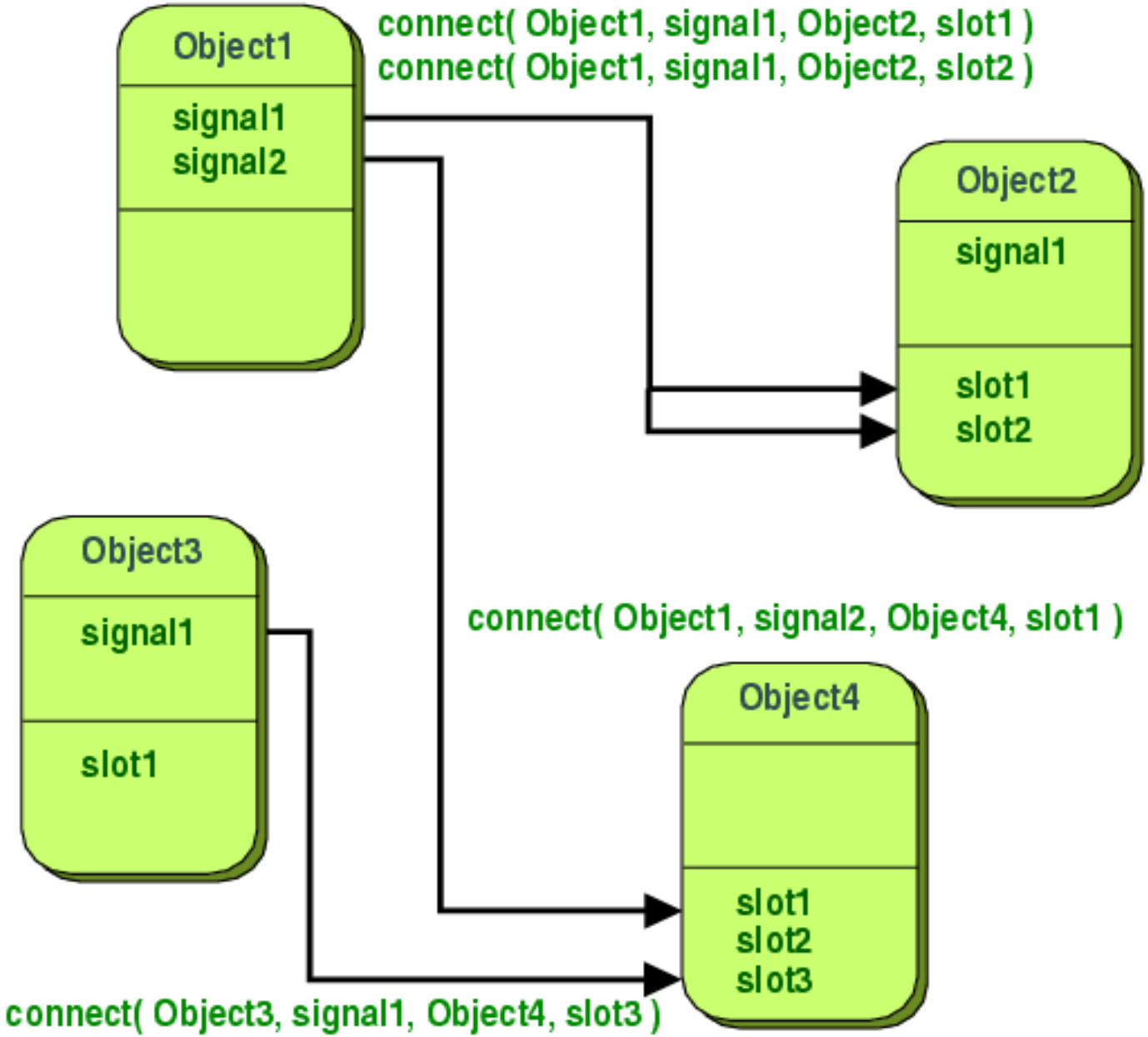



```
{
   ...
   emit clicked();
   ...
}
```

```
clear();
```



信号(Signal)与槽(Slot)初探

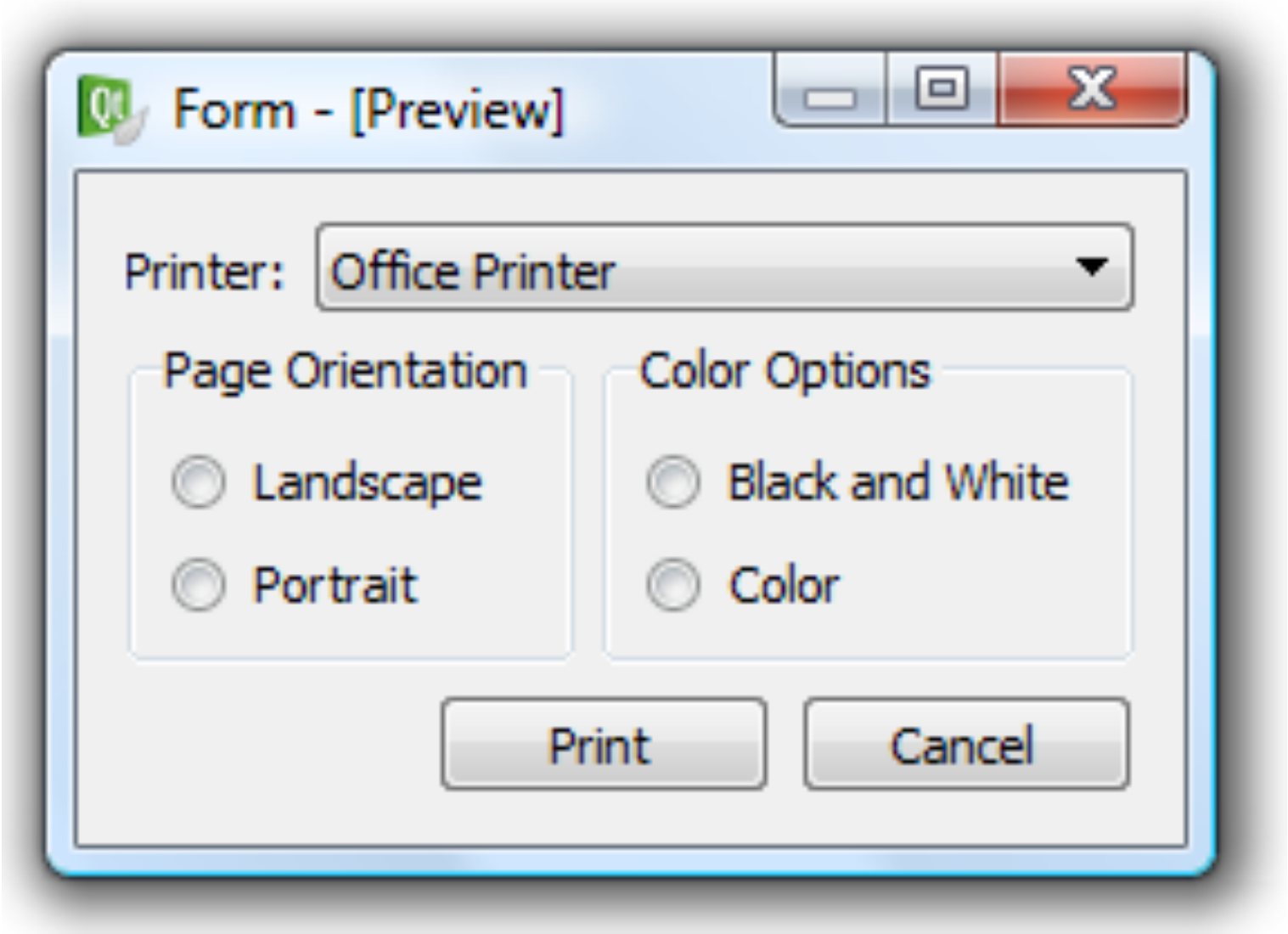


信号(Signal)与槽(Slot)初探

- 信号与槽机制只能用在继承于QObject的类
- 槽可以返回值，但通过联接返回时不能有返回值，槽以一个普通的函数实现，可以作为普通函数调用
- 信号总是返回空，信号总是不必实现
- 一个信号可以连接到多个槽，但槽的调用顺序不确定
- 信号和槽需要具有相同的参数列表
  - 如果信号的参数比槽多，那么多余的参数会被忽略
  - 如果参数列表不匹配，Qt会产生运行时错误信息

用户界面设计

用户界面由特定的部件（widget）构建



用户界面设计—三种方式

1.绝对定位(absolute positioning)

- 最粗劣的方式
- 对部件的大小、位置进行硬编码

2. 手工布局(manual layout)

- 绝对位置，但通过resizeEvent()方法改变大小

3.布局管理器(layout managers)

- 部件放置在布局管理器中，使界面更具弹性。

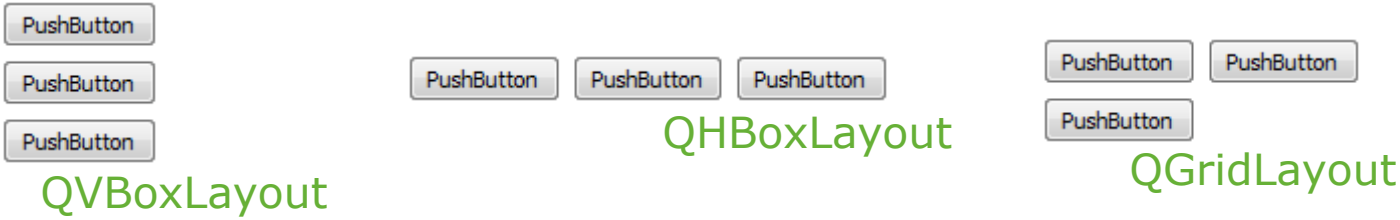
布局管理器的优点？

- 让部件适应内容

- 让部件适应翻译变化
- 让部件适应用户设置

布局管理

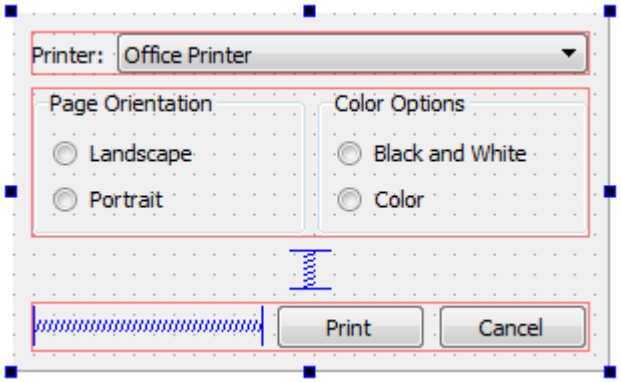
- 几种可用的布局



- 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 

布局管理器示例

- 对话框由多层的布局管理器和部件组成



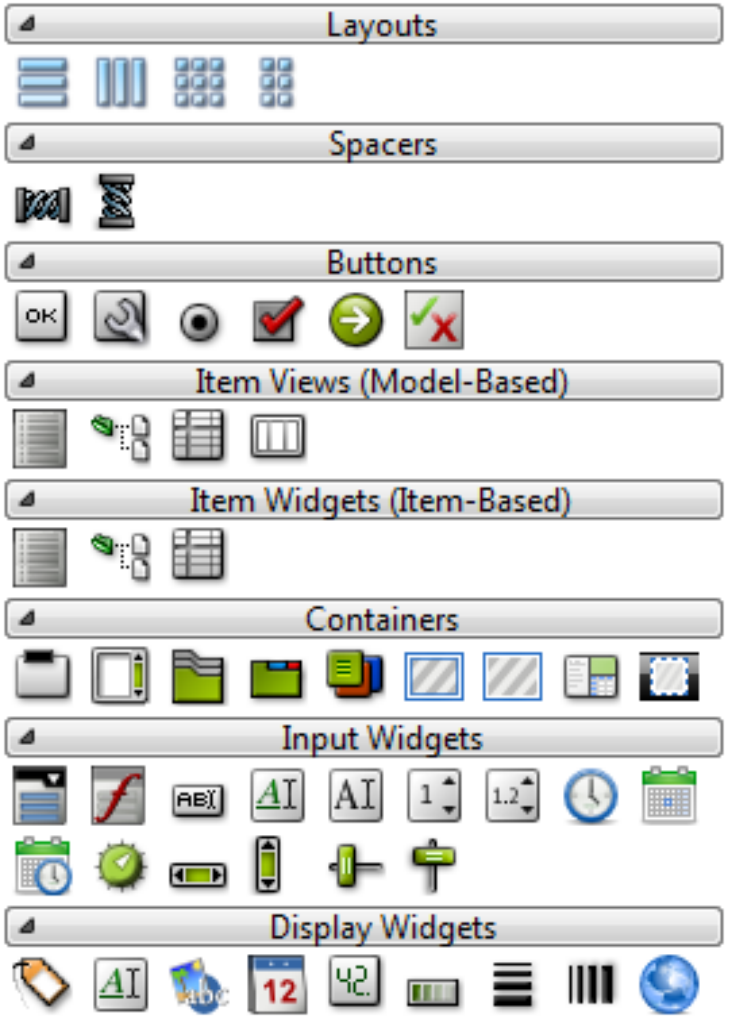
Object	Class
Form	QWidget
horizontalLayout	QHBoxLayout
label	QLabel
printerBox	QComboBox
horizontalLayout_2	QHBoxLayout
cancelButton	QPushButton
horizontalSpacer	Spacer
printButton	QPushButton
horizontalLayout_3	QHBoxLayout
groupBox	QGroupBox
landscapeButton	QRadioButton
portraitButton	QRadioButton
groupBox_2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

注意：布局管理器并不是其管理的部件的父对象

- 两种方式：代码实现，使用设计器

通用部件

- Qt包含针对所有情形的大量通用部件；
- 第三方控件，如QWT
- 自定义控件



尺寸（size）的策略

- 布局管理器是在空间和其他布局管理器之间进行协调
- 布局管理器提供布局结构
  - 水平布局和垂直布局
  - 网格布局
- 部件则提供
  - 各个方向上的尺寸策略
  - 最大和最小尺寸

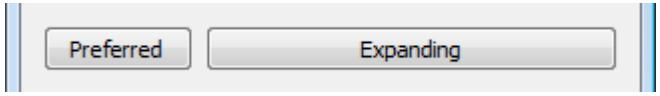
尺寸的策略

- 每一个widget有一个大小的示意，它给出了各个方向上尺寸的策略
  - Fixed –规定了widget的尺寸
  - Minimum – 规定了可能的最小值
  - Maximum – 规定可能的最大值
  - Preferred – 给出最好的值但不是必须的
  - 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?

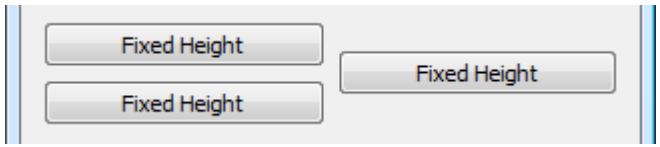
- 2个 preferred 相邻

- 1个 preferred, 1个 expanding



- 2个 expanding 相邻

- 空间不足以放置widget (fixed)



关于尺寸的更多信息

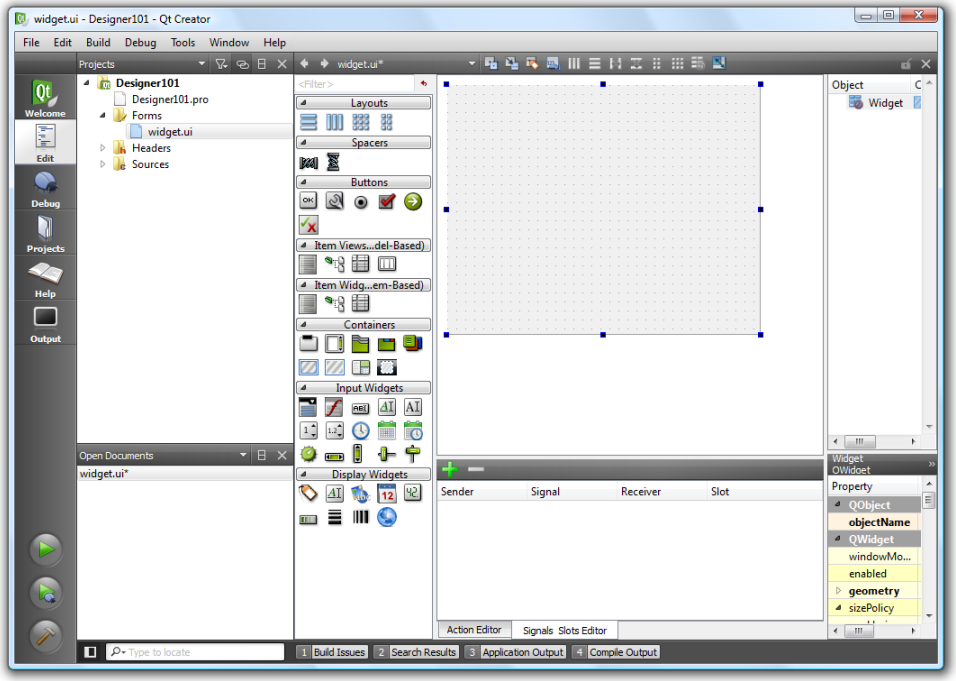
- 可用最大和最小属性更好地控制widget的大小

- maximumSize –最大可能尺寸
- minimumSize –最小可能尺寸

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

设计器介绍

- 以前设计器（Designer）是一个独立的工具，但现在是QtCreator的一个组成部分



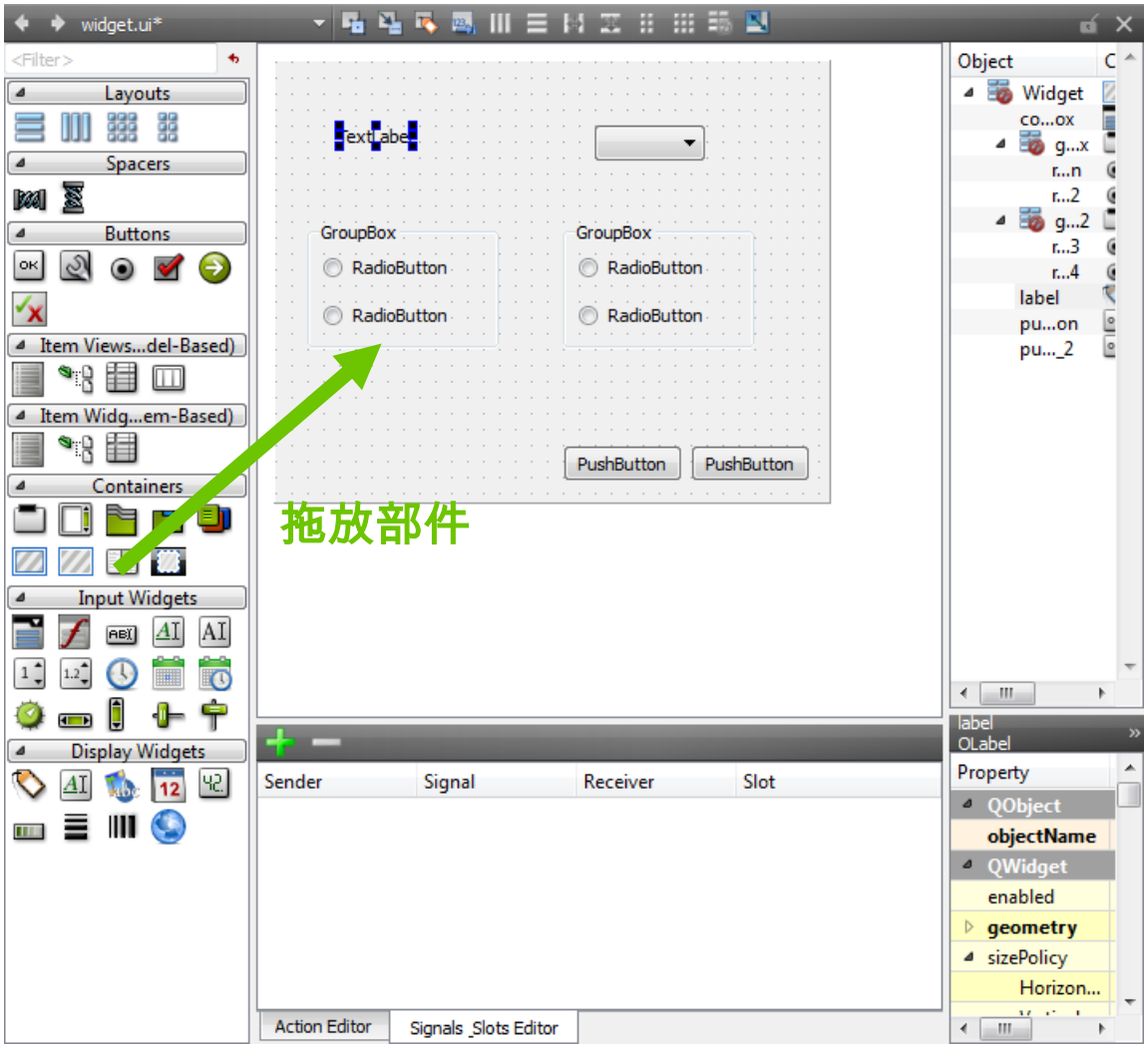
- 可视化窗体编辑器
- 拖放部件
- 安排布局
- 进行信号连接

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!

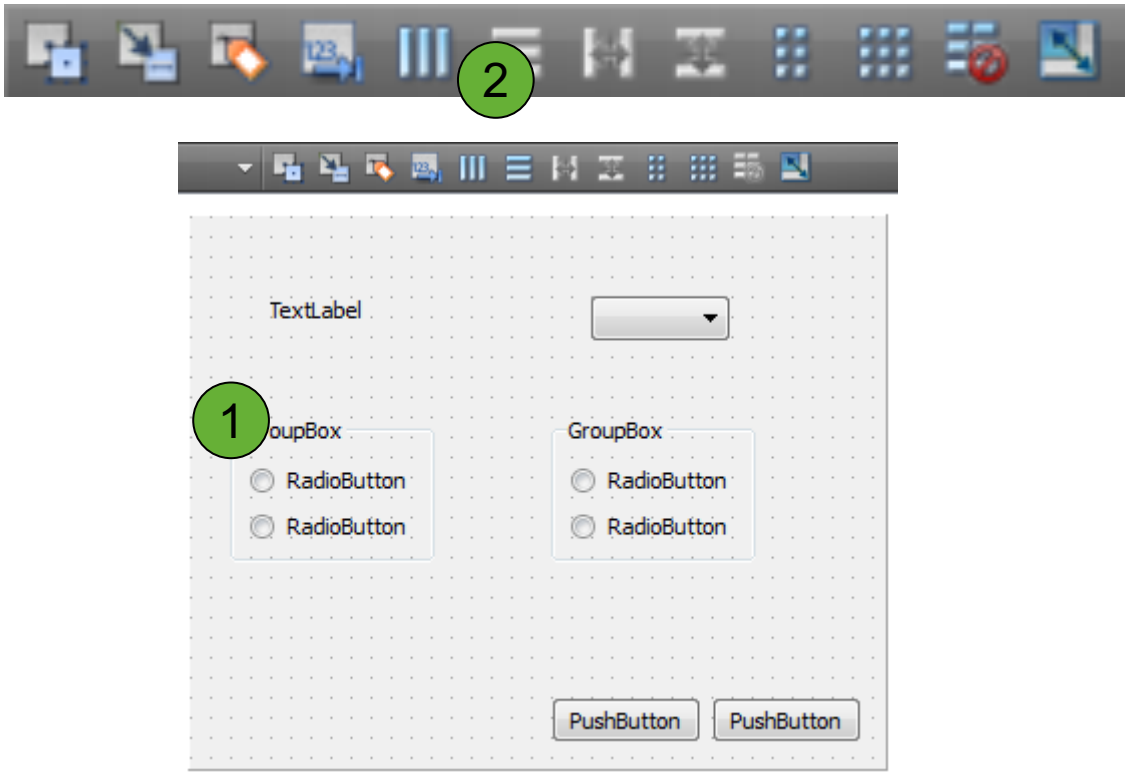
使用设计器

粗略地放置部件在窗体上



使用设计器

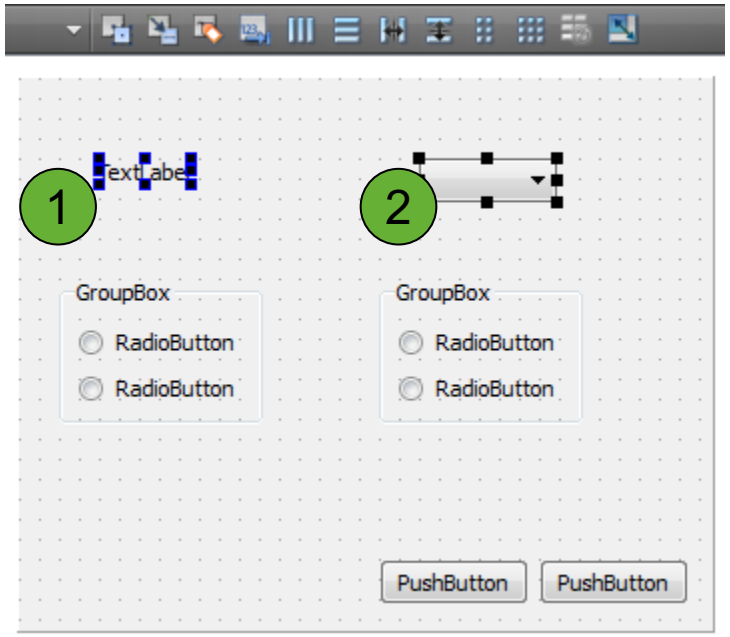
从里到外进行布局，添加必要的弹簧



1. 选中每一个 group box, **2**. 应用垂直布局管理

使用设计器

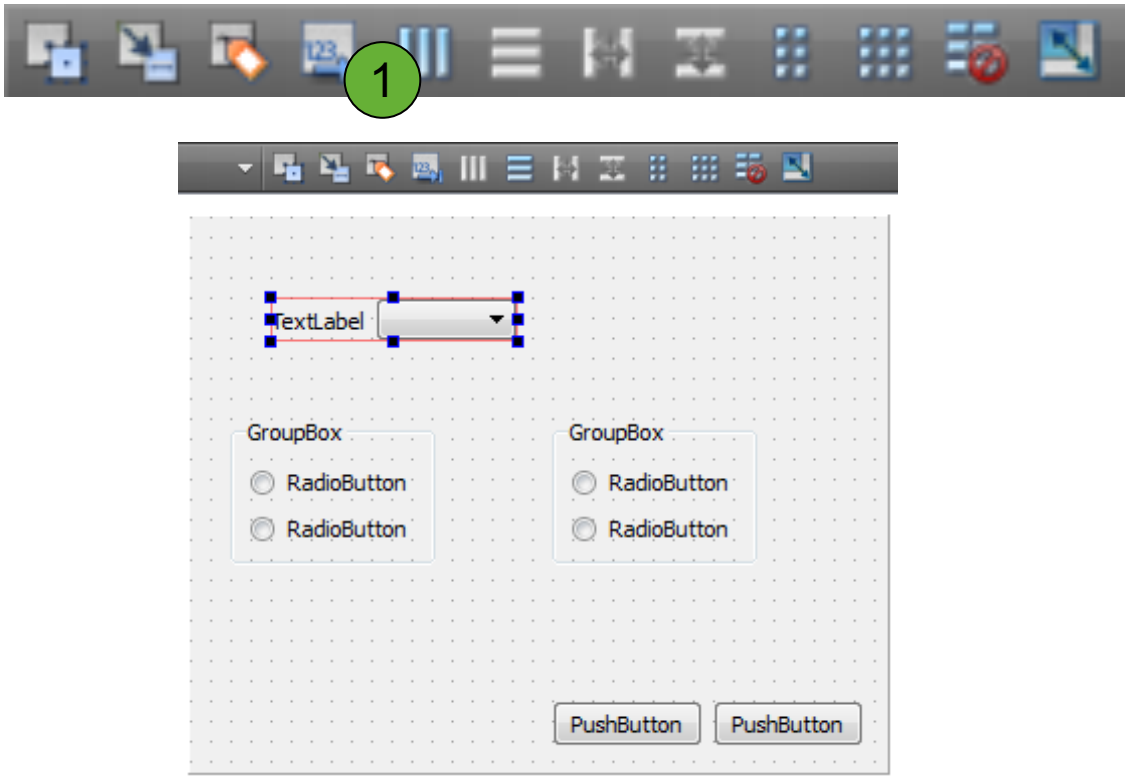
从里到外进行布局，添加必要的弹簧



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

使用设计器

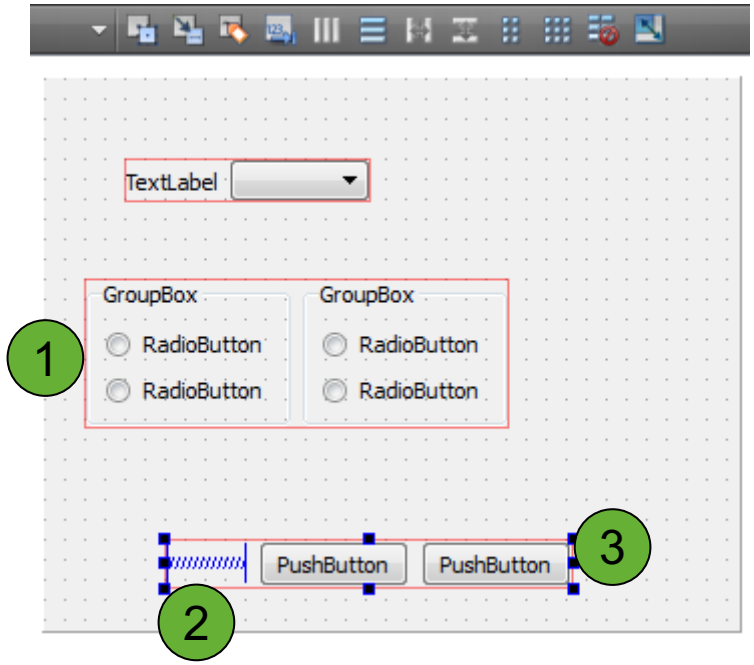
从里到外进行布局，添加必要的弹簧



1. 应用一个水平布局管理

使用设计器

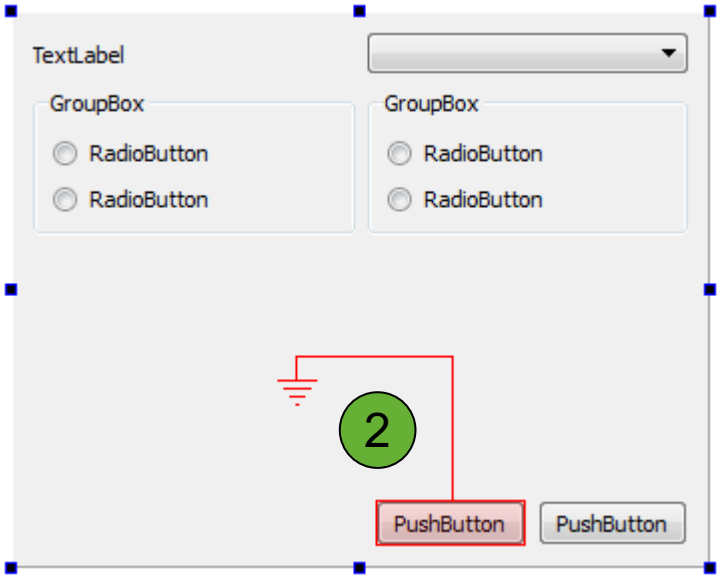
从里到外进行布局，添加必要的弹簧

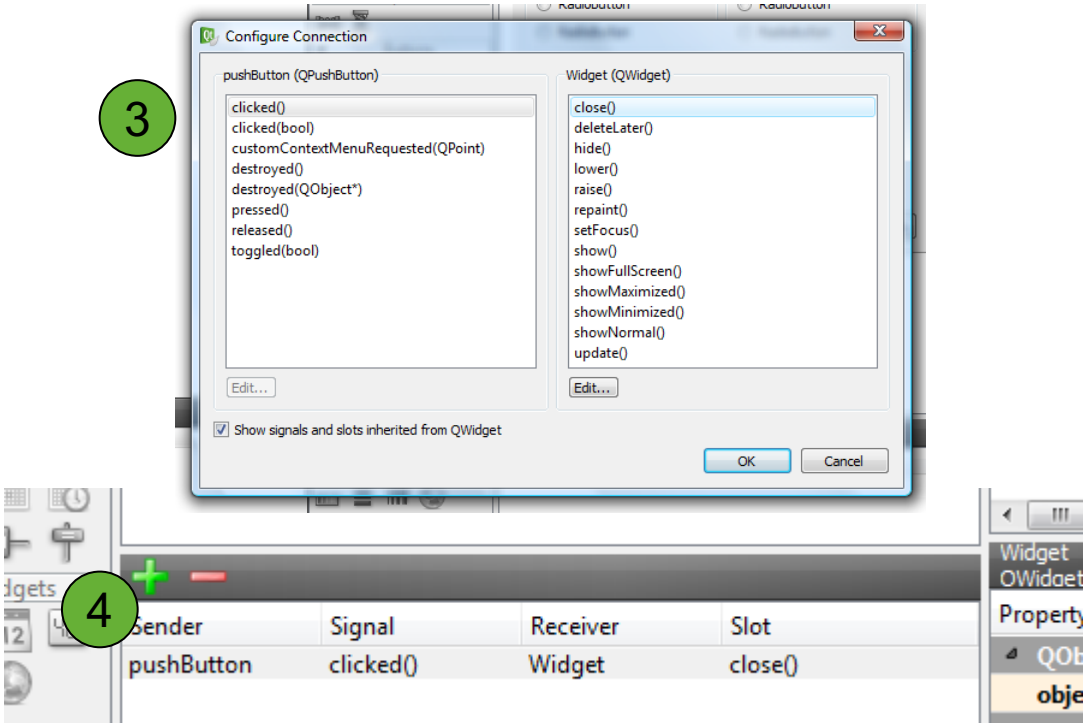


1. 选中2个group box并进行布局管理, **2**. 添加一个水平弹簧,
3. 将弹簧和按钮放置进一个布局管理中

使用设计器

进行信号连接(部件之间)

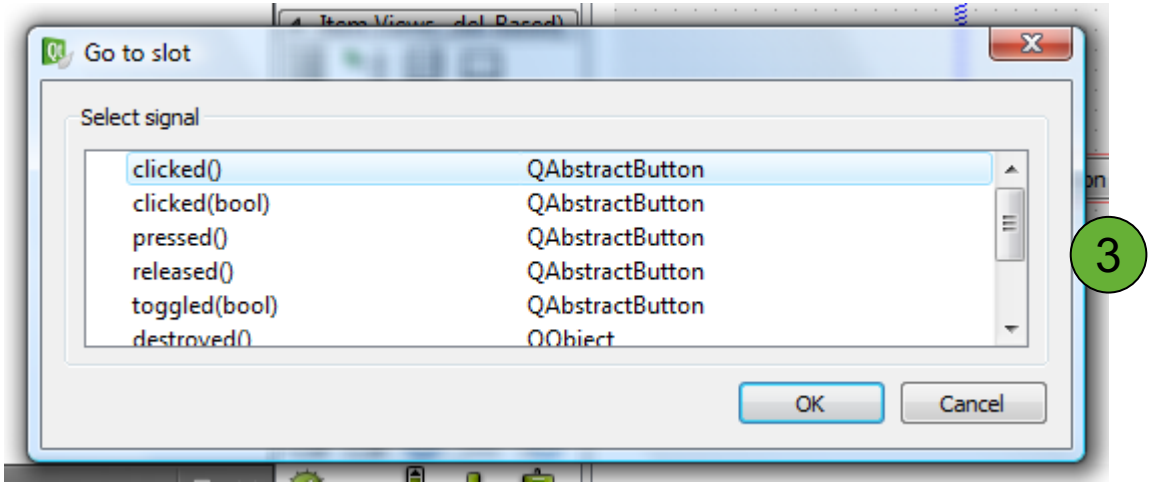


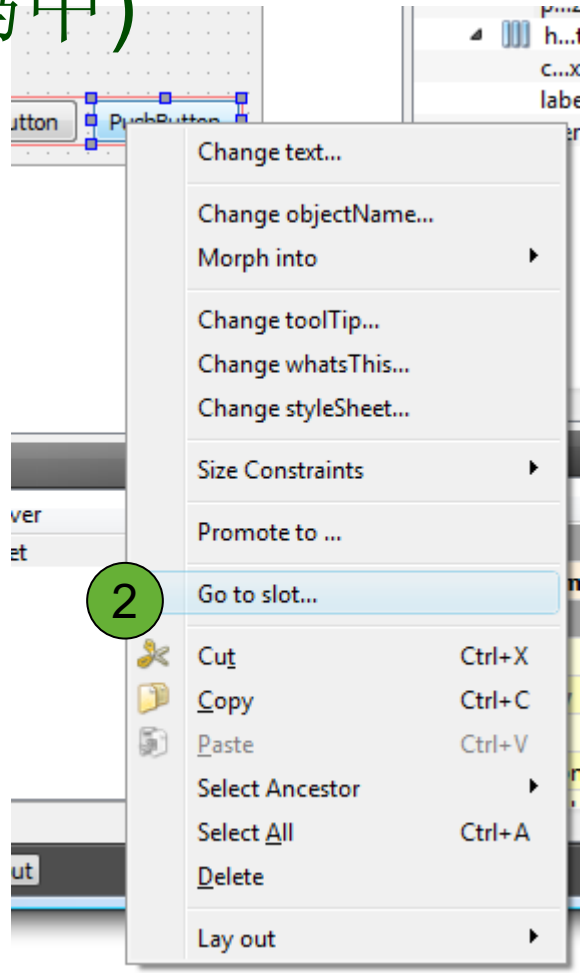


1. 转到signals and slot 编辑模式, **2**. 从一个部件拖放鼠标到另一个部件,
3. 选中signal and slot, **4**. 在connections‘ dock中查看结果

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类成员使用所有部件

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

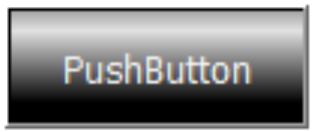
```
void Widget::memberFunction()
{
    ui->pushButton->setText(...);
}
```

样式表(StyleSheet)

- 所有的 QWidget 类都有一个 styleSheet 属性以支持跨平台样式
- 样式表是受启发自CSS的
- 它们可以用来进行高亮处理并进行许多小的修改

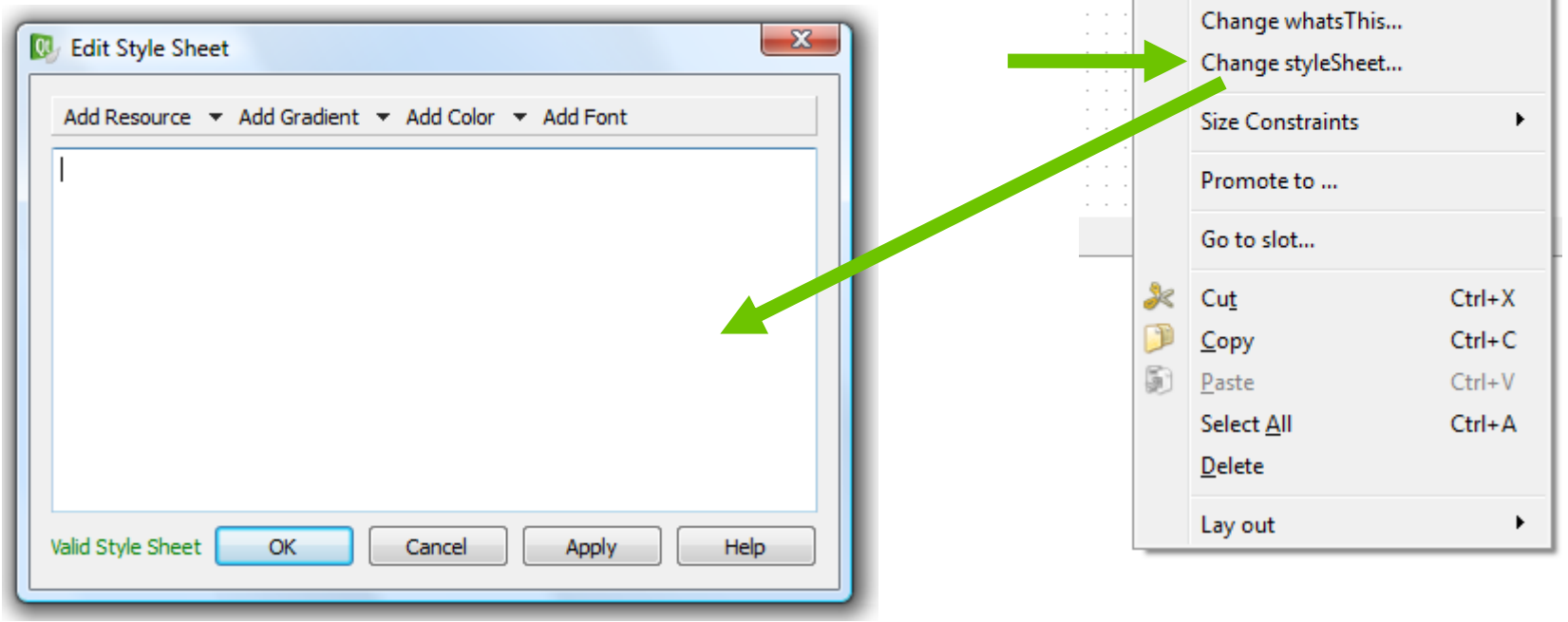
Hello World Hello World

- 当然也可以用于用户界面的整体修改



样式表

- 为一个单独的部件应用一个样式表的最简单方法是用设计器



资源文件(qrc)

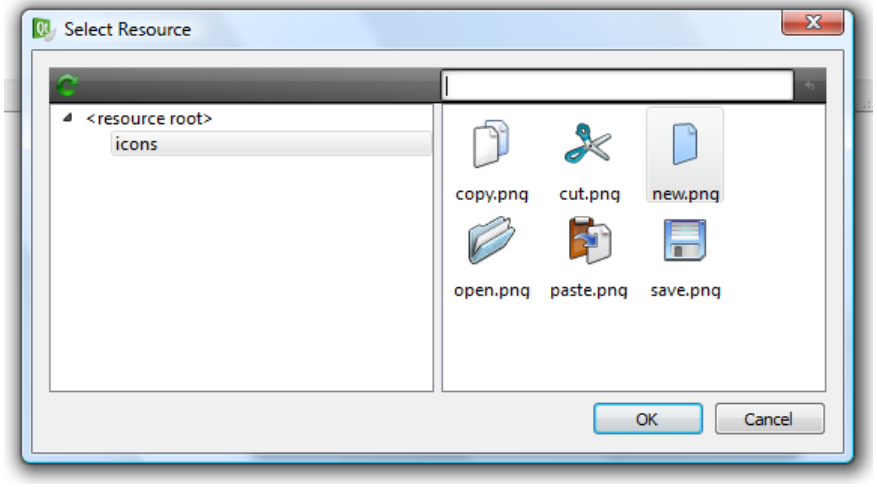
- 将图标放进一个资源文件中，Qt会将它们内嵌进可执行文件
  - 避免调用多文件
  - 不需要尝试确定每个特定安装风格下的图标的路径
  - 一切都巧妙地在软件构建系统中自适应
  - 避免部署的时候出现文件丢失的错误
  - 可以将任何东西添加进资源文件中，不仅仅是图标，但一般是不需要修改的文件。

资源文件(qrc)

- 可以轻松的在QtCreator中管理资源文件
- 在路径和文件名前添加：以使用资源

```
QPixmap pm(":/images/logo.png");
```

- 或者简单地在设计器的列表中选择一个图标



Qt的国际化

1. 确保应用程序是可翻译的：
   - 所用用户可见的字符串都使用tr()修饰
   - 根据不同的目标语言加载不同的qm的文件。
2. 即使应用程序目前不需要翻译，也应该为以后的需求留出余地。

Qt国际化—步骤

1.在代码中使用tr()修饰用户可见的字符串；

2.lupdate提取需要翻译的字符串；

```
TRANSLATIONS = spreadsheet_cn.ts \
               spreadsheet_en.ts
```

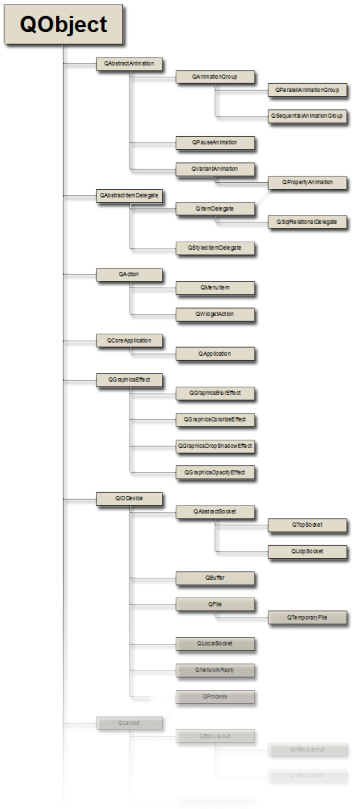
3.使用linguist工具翻译；

4.在程序开始时加载正确的qm文件。

- Qt简介
- Qt的使用
- Qt深入理解
- Qt的应用

QObject类

QObject是几乎所有Qt类和所有部件(widget)的基类。



它包含很多组成Qt的机制

- 事件
- 信号和槽
- 属性
- 内存管理

QObject类

QObject 是大部分Qt 类的基类

例外的例子是:

类需要作为轻量级的类，例如图元（graphical primitives）-QPen、QBrush。

数据容器(QString, QList, QChar等)

需要可复制的类，因为QObject类是无法被复制的。

QObject类

“QObject 的实例是单独的!”

它们可以拥有一个名字 (QObject::objectName)

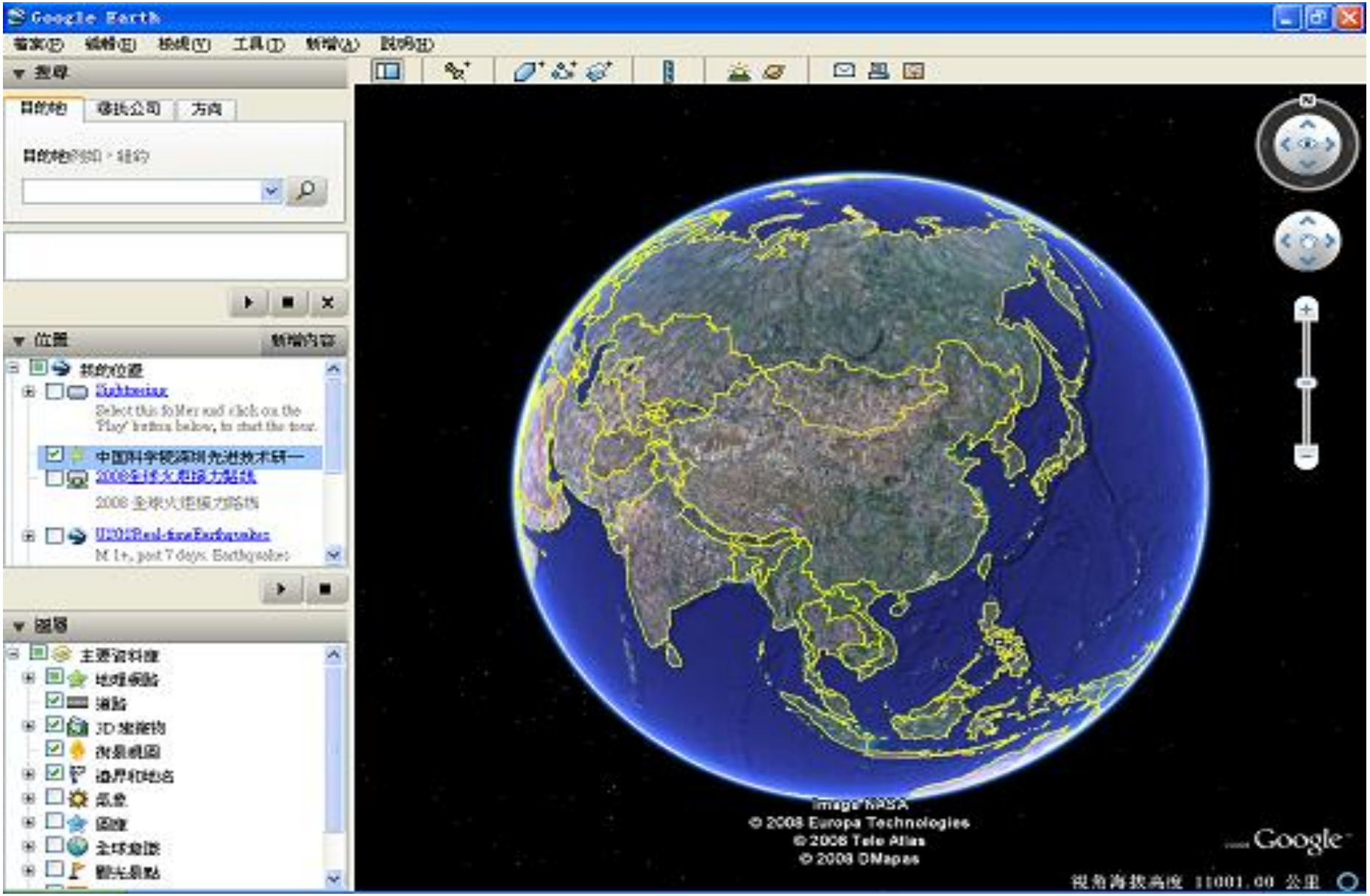
addButton, lineEdit_Password….

它们被放置在QObject实例的一个层次上

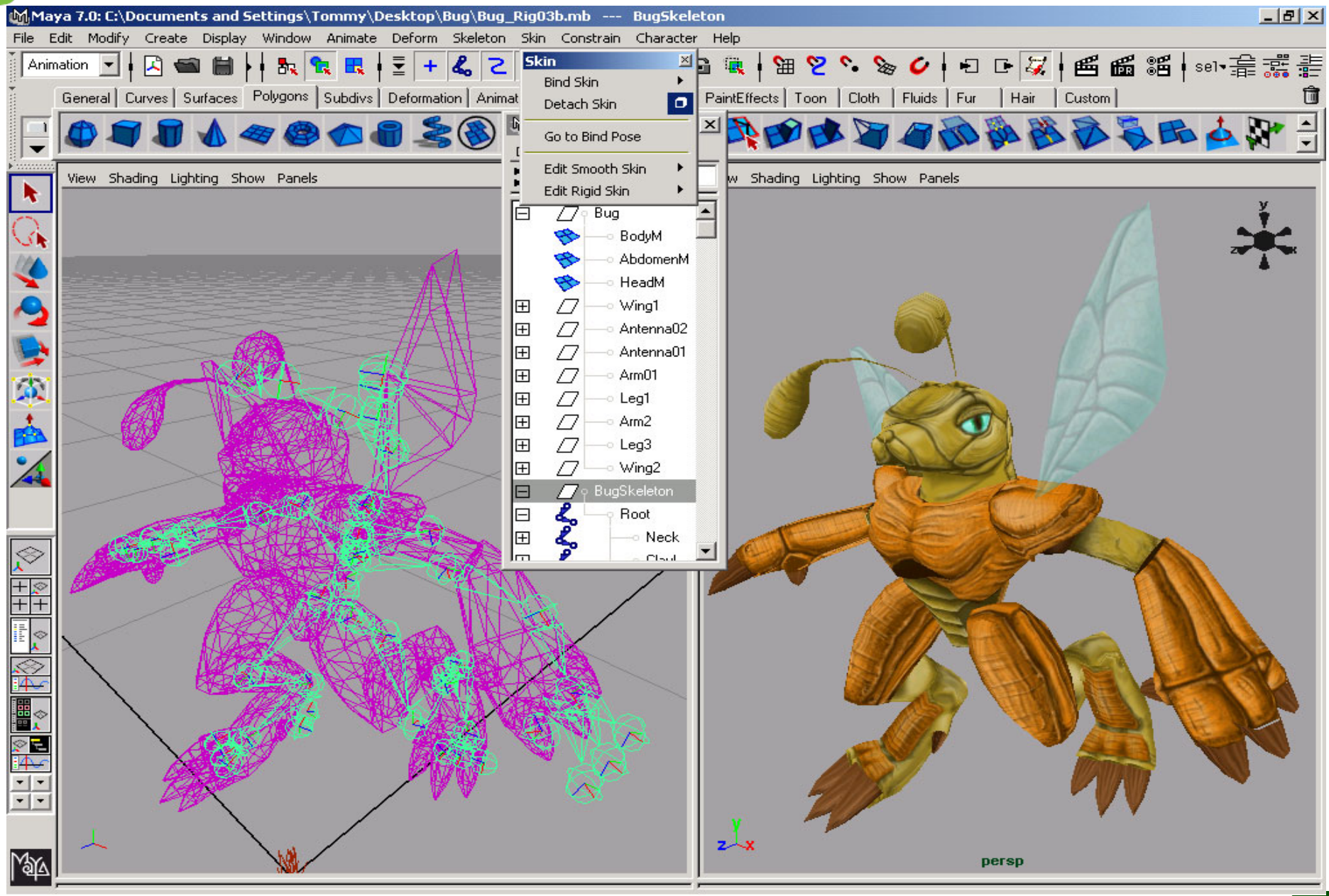
它们可以有到其他 QObject 实例的联接

- Qt简介
- Qt的使用
- Qt深入理解
- Qt的应用

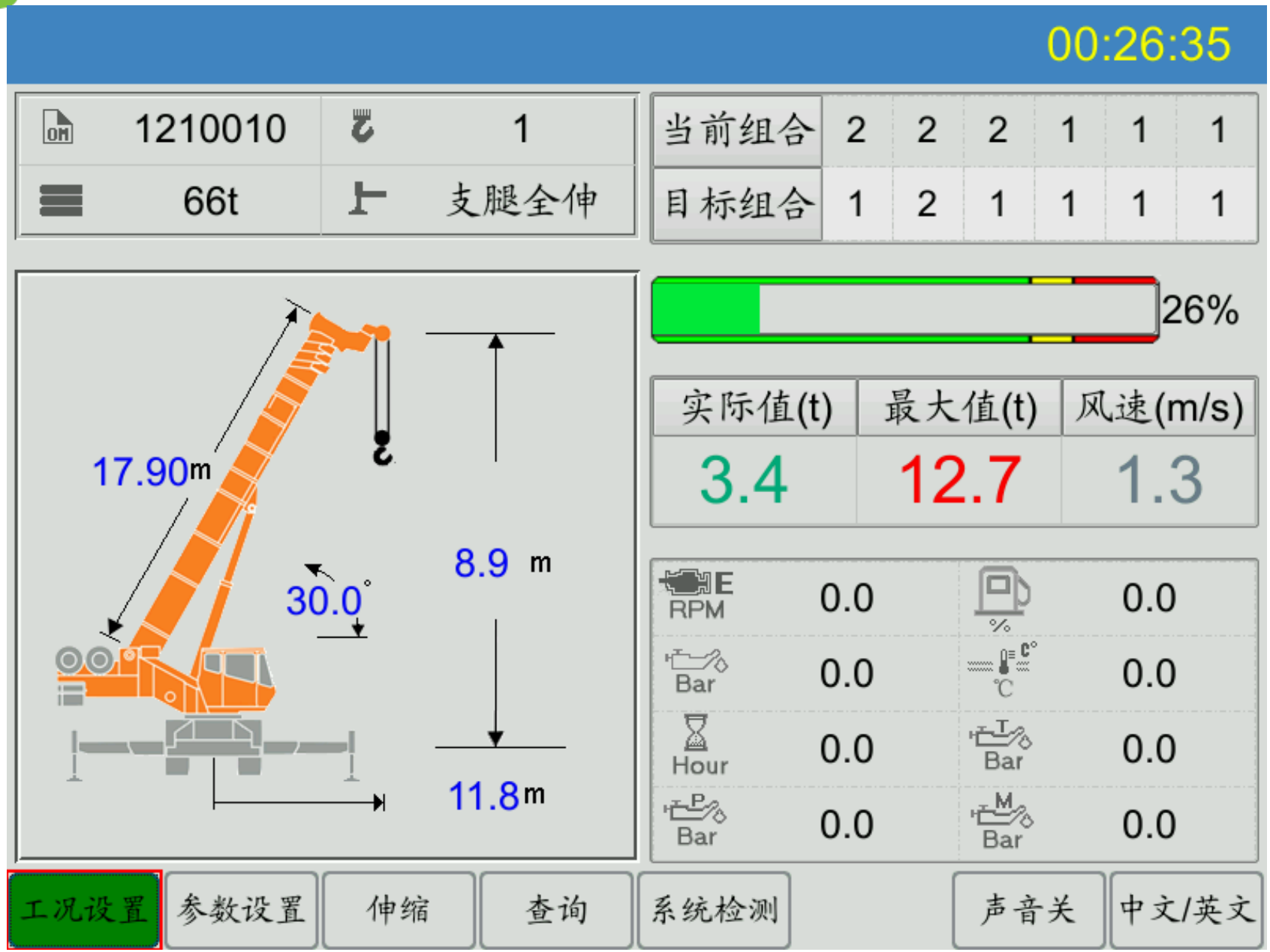
基于Qt开发的软件—Google Earth



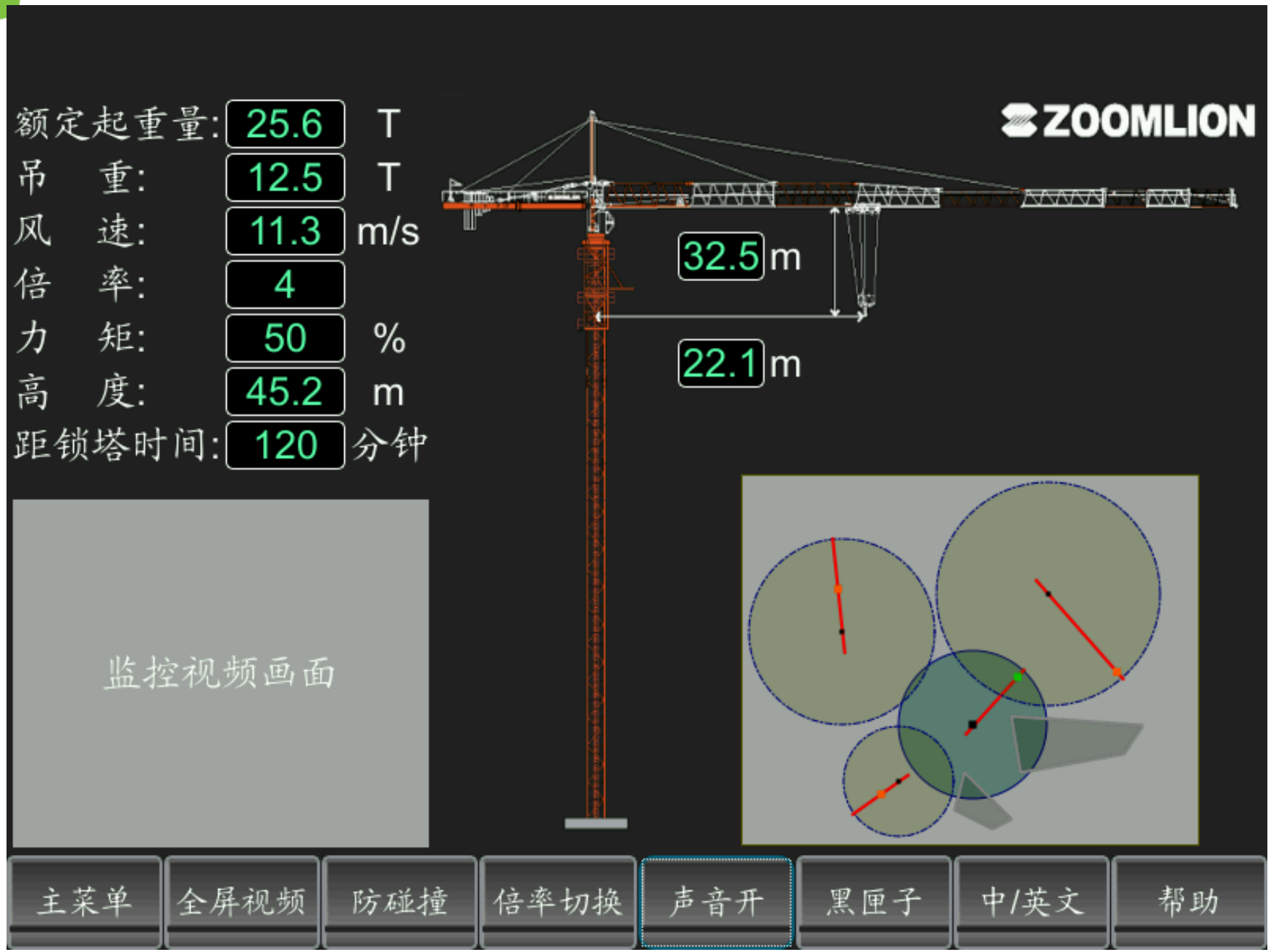
基于Qt开发的软件—MAYA



基于Qt开发的软件-大吨位力矩限制器



基于Qt开发的软件-塔机智能监控系统



谢谢！