Qt简介与信号/槽机制

课程主要内容

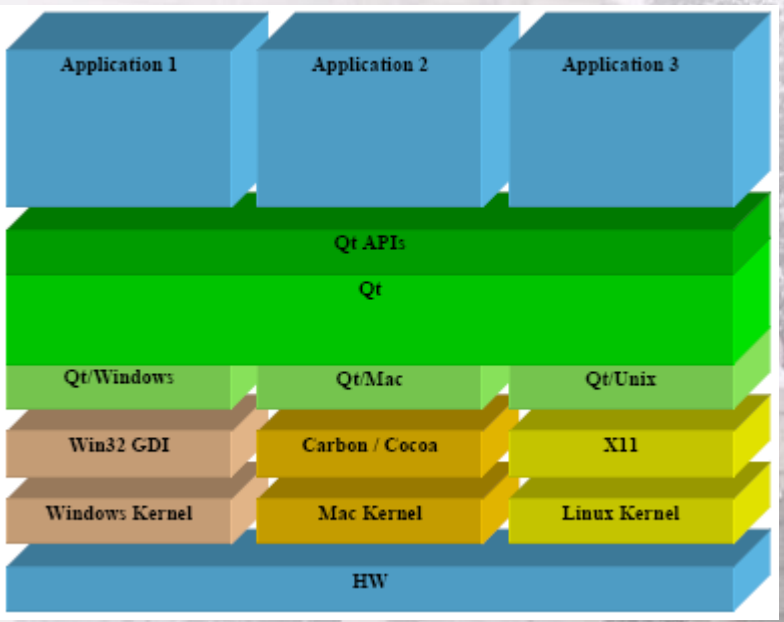
- **Qt是什么？**
- 应用举例
- **HelloWorld程序**
- 编译过程
- 应用程序执行过程
- 程序要素和主要基类
- 元数据与内省
- 对象树及内存管理
- 信号/槽机制
- 简单实例：温度转换器

清華大學

Qt是什么？

Qt是什么

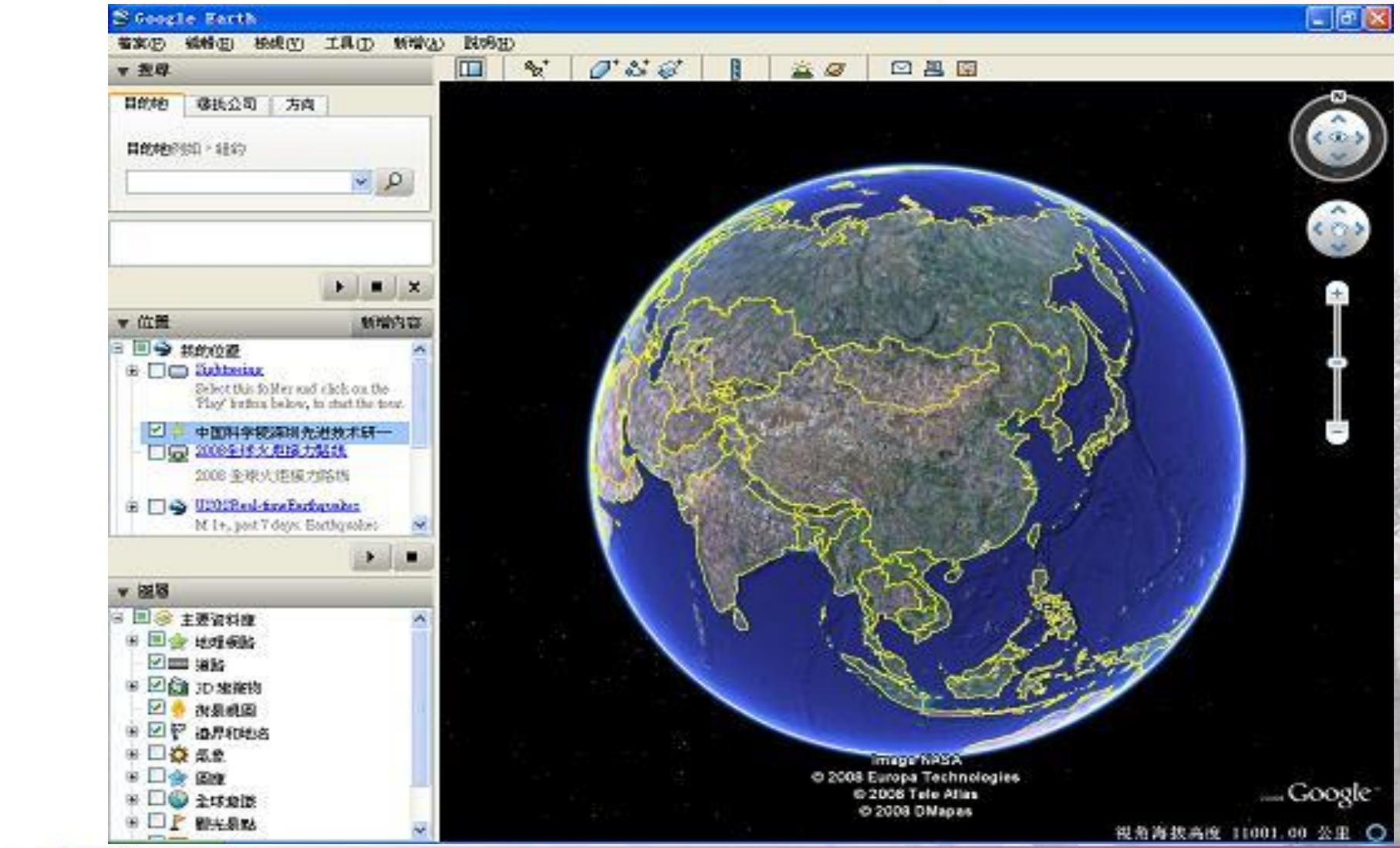
- Qt是Trolltech公司的标志性产品，后经Nokia公司，转至Digia公司，是跨平台的C++图形用户界面（GUI）工具包
- Qt 应用程序接口与工具兼容于所有支持平台，让开发员们掌握一个应用程序接口，便可执行与平台无关的应用开发与配置
  - Qt/Windows
  - Qt/Mac
  - Qt/X11 (Linux, Solaris, HP-UX, IRIX, AIX等)
  - Embeded Linux, Windows CE, Meego, Symbian
- Qt对不同平台的专门API进行了专门的封装（文件处理，网络等）



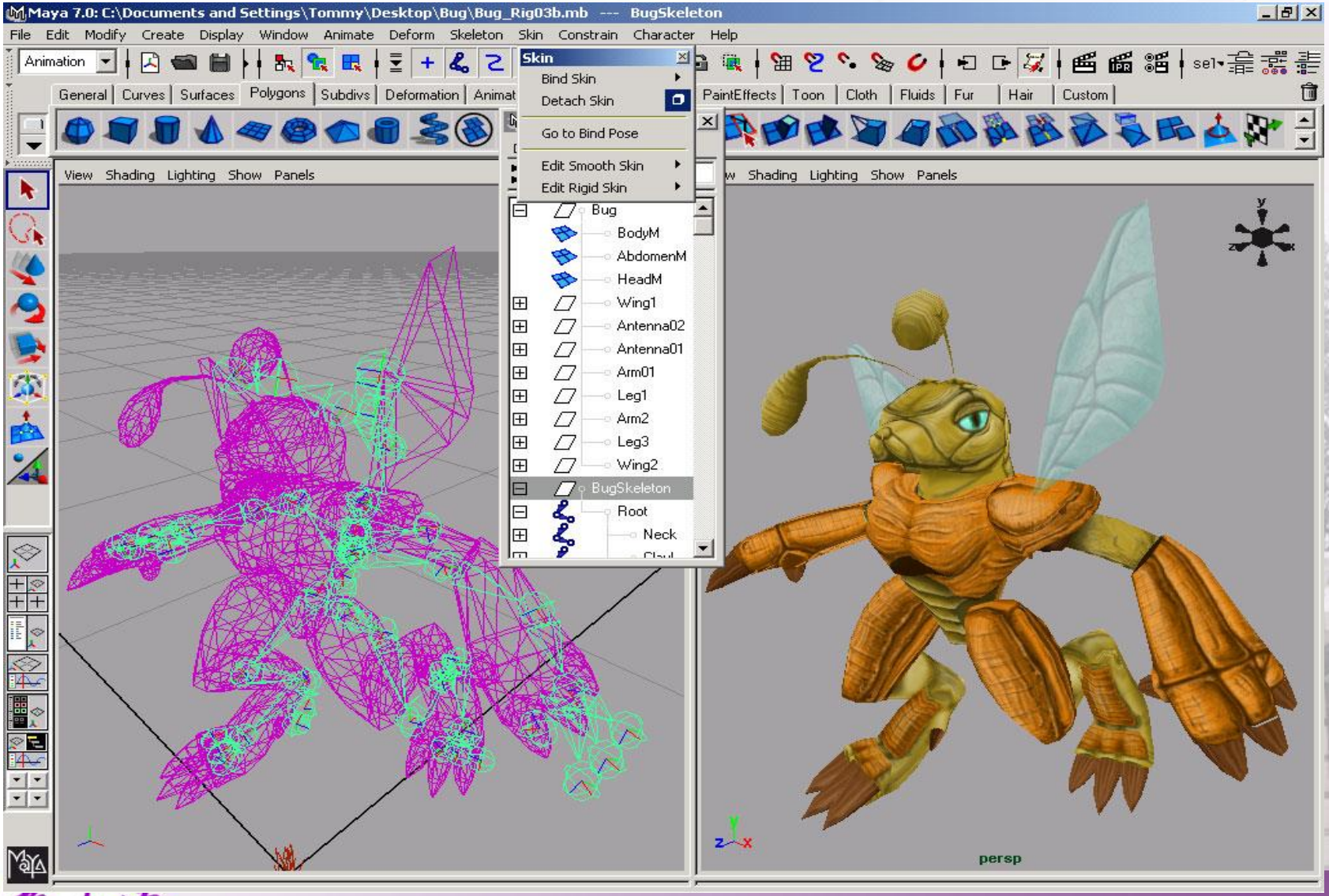
Qt应用举例

Qt开发的Linux桌面环境KDE

Qt开发的Google地球



Qt开发的三维动画软件MAYA



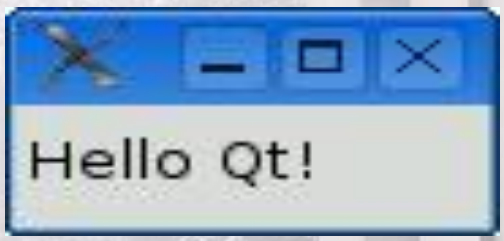
Qt开发的其他软件

- Opera浏览器
- Skype网络电话
- Linux下计算机辅助设计软件包QCad
- Adobe Photoshop Album
- CGAL计算几何库
- ……

HelloWorld

HELLO QT (1)

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QLabel *label = new QLabel("Hello Qt!");
    label->show();

    return app.exec();
}
```



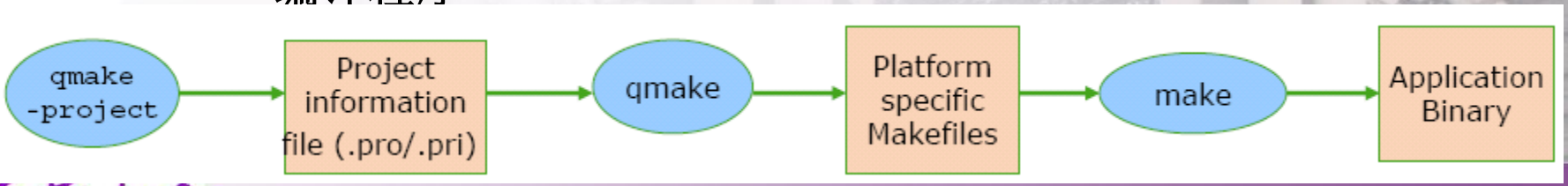
HELLO QT (2): 用HTML格式化


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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QLabel *label = new QLabel("<h2><i>Hello</i> "
                               "<font color=red>Qt!</font></h2>");
    label->show();

    return app.exec();
}
```

Qt编译过程

Qt编译过程

- 方法一：直接通过Qt IDE（Qt Creator）界面直接编译
- 方法二：命令行编译
  - 执行“qmake -project”
    - 创建 Qt 工程文件(.pro)，该工程文件也可以手动创建
  - 执行“qmake”
    - 缺省输入为工程文件，产生平台相关的 Makefile(s)
    - 产生编译规则，为工程中包含有 Q_OBJECT 宏的头文件调用 moc 编译器
  - 执行“make”
    - 编译程序

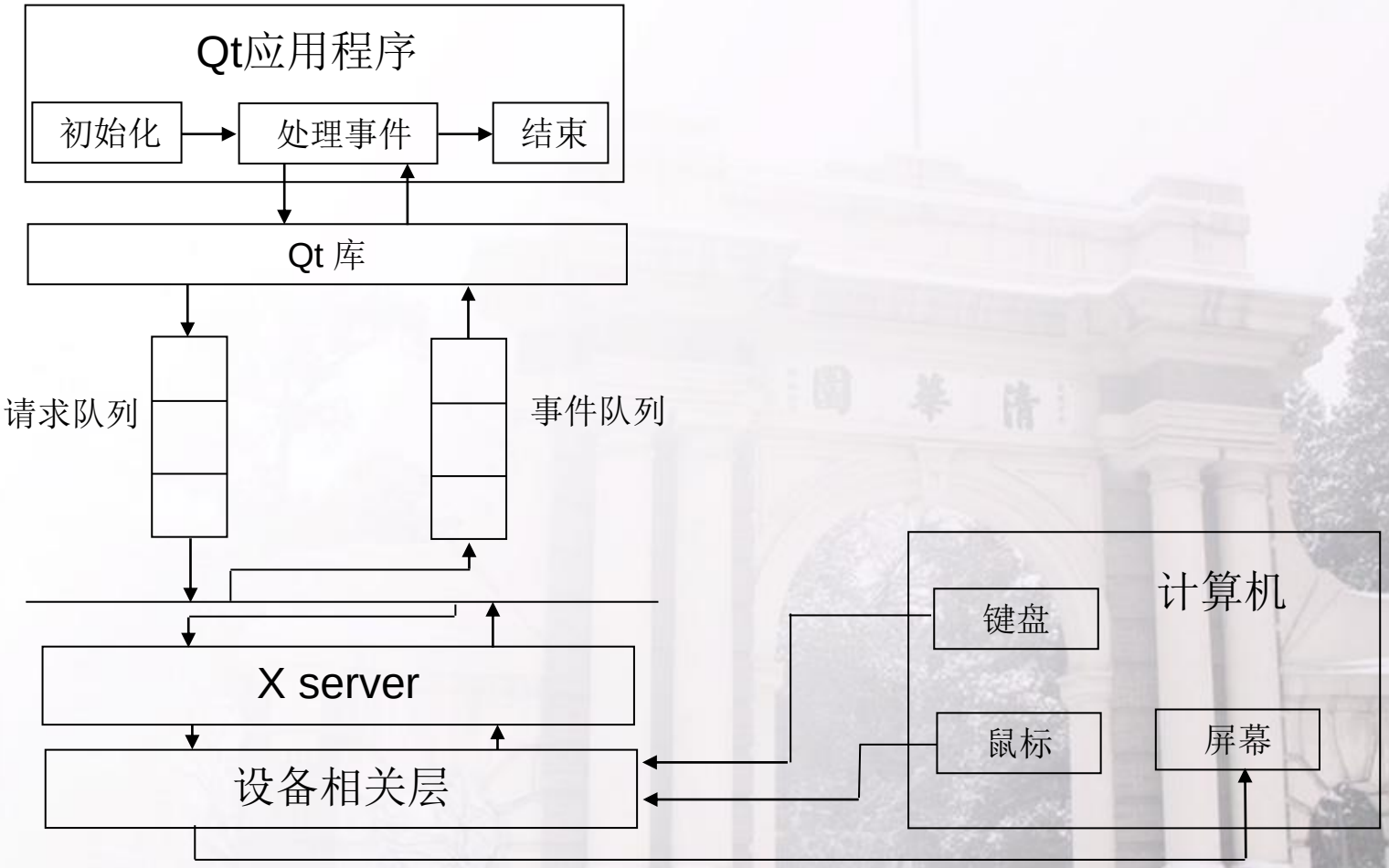


Qt编译工具：moc, uic 和 rcc

- moc，元对象（Meta-Object Compiler）编译器
  - 对每一个类的头文件，产生一个特殊的 meta-object
  - Meta-object 由 Qt 使用
- uic，Ui编译器
  - 根据Qt Designer产生的XML文件(.ui)生成对应的头文件代码
- rcc，资源编译器
  - 生成包含Qt资源文件(.qrc)中数据(如工具栏图标等)的C++源文件
- 这些工具在编译的时候由Makefile管理，自动运行

Qt应用程序执行过程

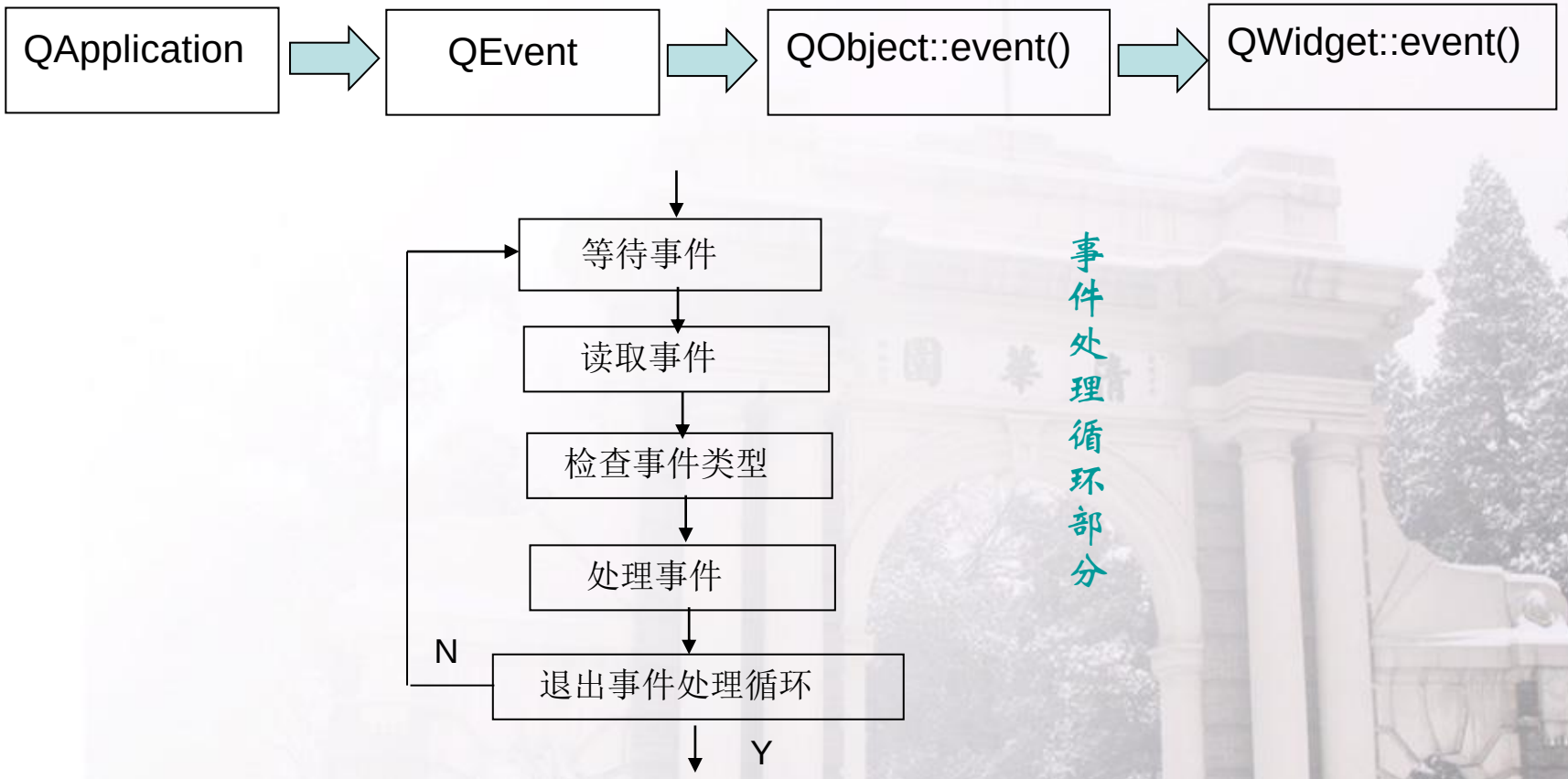
Qt应用程序执行过程—事件驱动



Qt事件处理

- 在X程序中，敲击键盘，鼠标指针在窗口中的移动或鼠标按键动作等，都是事件
- Qt事件的处理过程
  - QApplication的事件循环体从事件队列中拾取本地窗口系统事件或其他事件
  - 译成QEvent，并送给QObject::event()，最后送给QWidget::event()分别对事件处理

Qt事件处理（续）



如果在处理事件时，去执行耗时的计算、或阻塞的等待，会导致GUI响应慢，甚至僵死。

Qt程序要素和主要基类

主要基类：QObject类

- Qt对象模型的核心 – QObject类
  - QObject在整个Qt的体系中处于一个非常重要的位置
  - 是几乎所有Qt类和所有部件(widget)的基类
  - 所有的QWidgets都是QObject
  - 提供对象树和对象的关系
  - 提供了信号-槽的通信机制
  - 对象不允许拷贝（禁用拷贝构造函数）
- 包含了很多组成Qt的机制
  - 事件处理
  - 属性，内省（Introspection）
  - 内存管理

QObject类对象

- QObject类是所有能够处理signal、slot和事件的Qt对象的基类，原形如下：

```
QObject::QObject ( QObject * parent =0,const char * name = 0 )
```

- 创建带有父对象及其名字的对象，对象的父对象可以看作这个对象的所有者。比如，对话框是其中的ok和cancel按钮的父对象。
- 在上面的函数中如果parent为0则构造一个无父的对象，如果对象是一个组件，则它就会成为顶层的窗口。

QApplication类

- QApplication类负责GUI应用程序的控制流和主要的设置，包括：
  - 主事件循环体，负责处理和调度所有来自窗口系统和其他资源的事件
  - 处理应用程序的开始、结束以及会话管理
  - 还包括系统和应用程序方面的设置
- 在Qt应用程序中，首先要创建一个QApplication对象
  - 不管有多少个窗口，QApplication对象只能有一个，而且必须在其他对象之前创建
  - 可以利用全局指针qApp访问QApplication对象
- QApplication是QObject的子类

QApplication类

- QApplication类中封装很多函数，其中包括：
  - 系统设置：**setFont()**　用来设置字体
  - 事件处理：**sendEvent()**　用来发送事件
  - **GUI**风格：**setStyles()**　设置图形用户界面的风格
  - 颜色使用：**colorSpec()**　用来返回颜色文件
  - 文本处理：**translate()**　用来处理文本信息
  - 创建组件：**setmainWidget()**　用来设置窗口的主组件
  - **……**

函数分组	
系统设置	desktopSettingsAware()、setDesktopSettingsAware()、cursorFlashTime()、setCursorFlashTime()、doubleClickInterval()、setDoubleClickInterval()、wheelScrollLines()、setWheelScrollLines()、palette()、setPalette()、font()、setFont()、fontMetrics()。
事件处理	exec()、processEvents()、enter loop()、exit loop()、exit()、quit()。 sendEvent()、postEvent()、sendPostedEvents()、removePostedEvents()、hasPendingEvents()、notify()、macEventFilter()、qwsEventFilter()、x11EventFilter()、x11ProcessEvent()、winEventFilter()。
图形用户界面风格	style()、setStyle()、polish()。
颜色使用	colorSpec()、setColorSpec()、qwsSetCustomColors()。
文本处理	setDefaultCodec()、installTranslator()、removeTranslator()、translate()。
窗口部件	mainWidget()、setMainWidget()、allWidgets()、topLevelWidgets()、desktop()、activePopupWidget()、activeModalWidget()、clipboard()、focusWidget()、winFocus()、activeWindow()、widgetAt()。
高级光标处理	hasGlobalMouseTracking()、setGlobalMouseTracking()、overrideCursor()、setOverrideCursor()、restoreOverrideCursor()。
X窗口系统同步	flushX()、syncX()。
对话管理	isSessionRestored()、sessionId()、commitData()、saveState()。
线程	lock()、unlock()、locked()、tryLock()、wakeUpGuiThread()。
杂项	closeAllWindows()、startingUp()、closingDown()、type()。

程序退出

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

int main(int argc, char *argv[])
{
    QApplication app(argc, argv);

    QLabel *label = new
QLabel("Hello Qt!");
    label->show();

    return app.exec();
}
```

- 退出事件程序，只需要在程序结束时返回一个**exec()**，例如：

```
return  app.exec();
```

- 其中**app**为**QApplication**的实例，当调用**exec()**将进入主事件的循环中，直到**exit()**被调用或主窗口部件被销毁

QApplication类负责程序退出

- 退出应用程序可以调用继承自QCoreApplication类的quit或exit函数
  - QApplication是QCoreApplication类的子类
  - quit ()：告诉应用程序退出，并返回0（表示成功）
  - exit(0)：同quit()
- 举例

```
QPushButton *quitButton = new QPushButton("Quit");
connect(quitButton, SIGNAL(clicked()), qApp, SLOT(quit()));
```

或

```
qApp->exit(0);
```

QApplication类负责关闭窗口

- 调用QApplication::closeAllWindows ()
  - 尤其适用于有多个顶层窗口的应用程序
  - 如果关闭窗口后，不想让应用程序退出，则需要调用函数 QApplication::setQuitOnLastWindowClosed (false)
- 举例

```
exitAct = new QAction(tr("E&xit"), this);
exitAct->setShortcuts(QKeySequence::Quit);
exitAct->setStatusTip(tr("Exit the application"));
connect(exitAct, SIGNAL(triggered()), qApp, SLOT(closeAllWindows()));
```

QWidget类负责窗口部件

- QWidget类是所有用户界面对象的基类，是QObject类的子类，继承了QObject类的属性。
- 窗口部件是用户界面的一个原子：它从窗口系统接收鼠标、键盘和其它事件，并且在屏幕上绘制自己的表现。
- 按钮（Button）、菜单（menu）、滚动条（scroll bars）和框架（frame）都是窗口部件的例子。
- 通常，应用程序都是一个控件，只是这个控件是由很多其它的控件组成

QWidget类（续）

- 窗口部件可以包含其它的窗口部件。例如，一个应用程序界面通常就是一个包含了QMenuBar,一些QToolBar，一个QStatusBar和其它的一些部件的窗口。
- 绝大多数应用程序使用一个QMainWindow或者一个QDialog作为程序界面，但是Qt允许任何窗口部件成为窗口。
- 当窗口部件被创建的时候，它总是隐藏的，必须调用show()来使它可见。
- QWidget类有很多成员函数，但一般不直接使用，而是通过子类继承来使用其函数功能。如，QPushButton、QlistBox等都是它的子类

上下文	函数
窗口函数	show()、hide()、raise()、lower()、close()。
顶级窗口	caption()、setCaption()、icon()、setIcon()、iconText()、setIconText()、isActiveWindow()、setActiveWindow()、showMinimized()、showMaximized()、showFullScreen()、showNormal()。
窗口内容	update()、repaint()、erase()、scroll()、updateMask()。
几何形状	pos()、size()、rect()、x()、y()、width()、height()、sizePolicy()、setSizePolicy()、sizeHint()、updateGeometry()、layout()、move()、resize()、setGeometry()、frameGeometry()、geometry()、childrenRect()、adjustSize()、mapFromGlobal()、mapFromParent()、mapToGlobal()、mapToParent()、maximumSize()、minimumSize()、sizeIncrement()、setMaximumSize()、setMinimumSize()、setSizeIncrement()、setBaseSize()、setFixedSize()。
模式	isVisible()、isVisibleTo()、visibleRect()、isMinimized()、isDesktop()、isEnabled()、isEnabledTo()、isModal()、isPopup()、isTopLevel()、setEnabled()、hasMouseTracking()、setMouseTracking()、isUpdatesEnabled()、setUpdatesEnabled()。
观感	style()、setStyle()、cursor()、setCursor()、font()、setFont()、palette()、setPalette()、backgroundMode()、setBackgroundMode()、colorGroup()、fontMetrics()、fontInfo()。
键盘焦点函数	isFocusEnabled()、setFocusPolicy()、focusPolicy()、hasFocus()、setFocus()、clearFocus()、setTabOrder()、setFocusProxy()。
鼠标和键盘捕获	grabMouse()、releaseMouse()、grabKeyboard()、releaseKeyboard()、mouseGrabber()、keyboardGrabber()。
事件处理器	event()、mousePressEvent()、mouseReleaseEvent()、mouseDoubleClickEvent()、mouseMoveEvent()、keyPressEvent()、keyReleaseEvent()、focusInEvent()、focusOutEvent()、wheelEvent()、enterEvent()、leaveEvent()、paintEvent()、moveEvent()、resizeEvent()、closeEvent()、dragEnterEvent()、dragMoveEvent()、dragLeaveEvent()、dropEvent()、childEvent()、showEvent()、hideEvent()、customEvent()。
变化处理器	enabledChange()、fontChange()、paletteChange()、styleChange()、windowActivationChange()。
系统函数	parentWidget()、topLevelWidget()、reparent()、polish()、winId()、find()、metric()。
这是什么的帮助	customWhatsThis()。
内部核心函数	focusNextPrevChild()、wmapper()、clearWFlags()、getWFlags()、setWFlags()、testWFlags()。

QMainWindow类负责窗口创建

- 在Qt程序中，创建窗口比较简单，只要在main.cpp文件加入如下两行：
  - MainWindow w;
  - w.show();
- MainWindow是一个用户自定义的类，它是QMainWindow的子类
  - QMainWindow类是QWidget的子类，用于创建带有菜单栏和工具栏的窗口，如windows系统的浏览器
  - 另外，有QDialog类也是QWidget的子类，与QMainWindow不同的是，QDialog类用于创建对话框窗口，如多数软件都有的“关于”对话框。

Qt元数据与内省

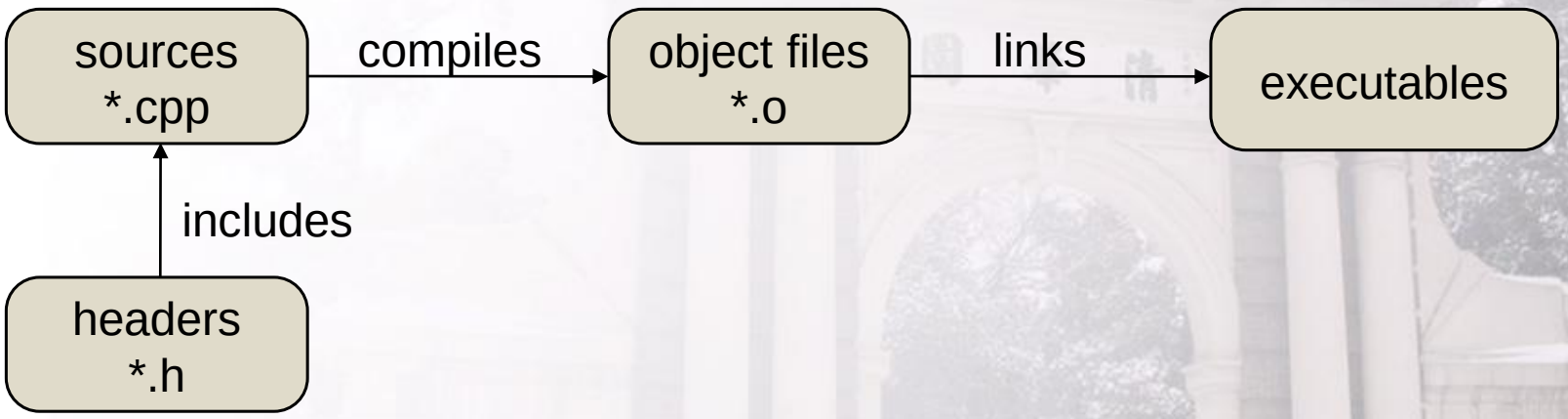
元数据（Meta data）

- Qt用C++实现内省（Introspection）
- 每一个 QObject 子类的对象都有一个元对象
- 元对象涉及：
  - 类名 (QObject::className)
  - 继承 (QObject::inherits)
  - 属性
  - 信号和槽
  - 普通信息(QObject::classInfo)

元数据

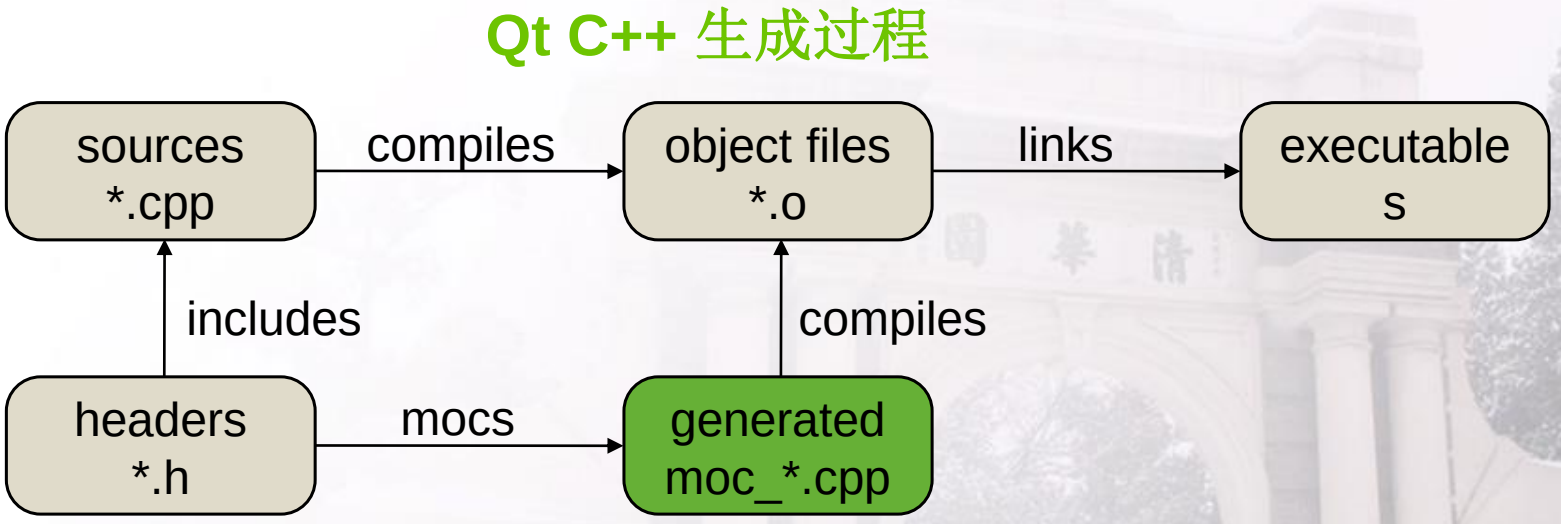
- 元数据通过元对象编译器(moc)在编译时组合在一起。

普通的C++生成过程



元数据Meta data

- 元数据通过元对象编译器(moc)在编译时组合在一起。



- **moc从头文件里面获得数据。**

元数据

- **moc 找什么？**

首先确认该类继承自 QObject（可能是间接）

Q_OBJECT 宏，通常是第一步

类的一般信息

Qt 关键字

```
class MyClass : public QObject
{
    Q_OBJECT
    Q_CLASSINFO("author", "John Doe")

public:
    MyClass(const Foo &foo, QObject *parent=0);

    Foo foo() const;

public slots:
    void setFoo( const Foo &foo );

signals:
    void fooChanged( Foo );

private:
    Foo m_foo;
};
```

内省(Introspection)

```
QTimer *timer = new QTimer;         // QTimer inherits QObject
timer->inherits("QTimer");          // returns true
timer->inherits("QObject");         // returns true
timer->inherits("QAbstractButton"); // returns false
```

元对象了解对象细节信息

```
const QMetaObject* metaObject = obj->metaObject();
QStringList methods;
for(int i = metaObject->methodOffset();
        i < metaObject->methodCount();
        ++i)
  methods << QString::fromLatin1(metaObject->method(i).methodSignature());
```

Qt对象树及内存管理

QObject类（续）--父子关系

- 每一个QObject对象都可以有一个指向父亲的参数
- 孩子会通知他的父亲自己的存在，父亲会把它加入到自己的孩子列表中
- 如果一个widget对象没有父亲，那么他就是一个窗口
- 父部件可以:
  - 当父部件隐藏或显示自己的时候，会自动的隐藏和显示子部件
  - 当父部件enable和disable时，子部件的状态也随之变化
- 注意：在父部件可见的时候，子部件也可以单独隐藏自己

改变所有者

- **QObject子类对象可以修改它所属的父对象。**

```
obj->setParent(newParent);
```

- **父对象知道何时子对象被删除**

```
delete listWidget->item(0); // 删除第一个item(不安全)
```

- **一系列函数实现返回指针，从其所有者“拿走”释放的数据，把它留给拿取者处理**

```
QLayoutItem *QLayout::takeAt(int);
QListWidgetItem *QListWidget::takeItem(int);

// Safe alternative
QListWidgetItem *item = listWidget->takeItem(0);
if (item) { delete item; }
```

item列表本质上并不是子对象，而是拥有者。

这个例子进行了说明。

QObject类（续）--内存管理

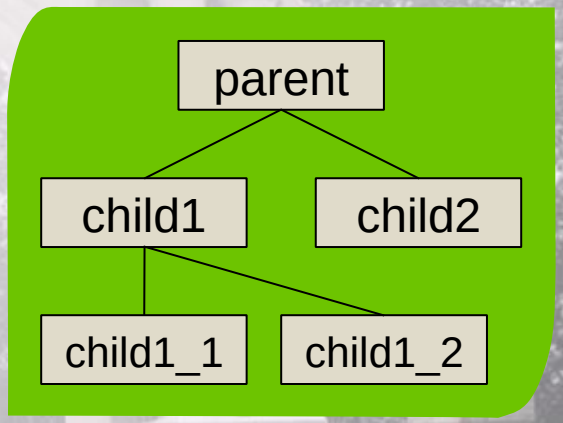
- 所有子对象的内存管理都转移给了父对象
  - 使用new在堆上分配内存
  - 子对象可自动被父对象删除内存
  - 手动删除不会引起二次删除，因为子对象删除时会通知父对象
- 没有父对象的QObject对象都需要手动删除
  - 一般把这种无父亲的对象分配在栈上，可以避免内存泄露的问题
- Qt是否有类似于自动回收站的机制？但是事实是没有的!
  - 只需要关注对象的父子关系和功能!

内存管理

- **QObject** 可以有父对象和子对象
- 当一个父对象被删除，它的子对象也同样被删除。

```
QObject *parent = new QObject();
QObject *child1 = new QObject(parent);
QObject *child2 = new QObject(parent);
QObject *child1_1 = new QObject(child1);
QObject *child1_2 = new QObject(child1);

delete parent;
```

parent 删除 child1 和 child2
child1 删除 child1_1 和 child1_2

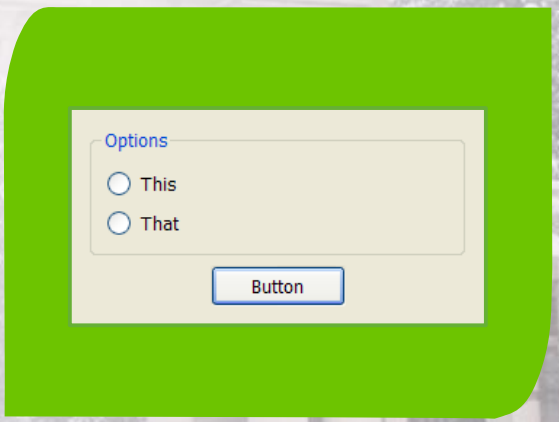


内存管理

- 当需要实现视觉层级时使用到它

```
QDialog *parent = new QDialog();
QGroupBox *box = new QGroupBox(parent);
QPushButton *button = new QPushButton(parent);
QRadioButton *option1 = new QRadioButton(box);
QRadioButton *option2 = new QRadioButton(box);

delete parent;
```

parent 删除 box 和 button
box 删除 option1 和 option2



使用模式

- 使用 **this**指针指向最高层父对象

```
Dialog::Dialog(QWidget *parent) : QDialog(parent)
{
    QGroupBox *box = QGroupBox(this);
    QPushButton *button = QPushButton(this);
    QRadioButton *option1 = QRadioButton(box);
    QRadioButton *option2 = QRadioButton(box);
    ...
```

- 在栈上分配父对象空间

```
void Widget::showDialog()
{
    Dialog dialog;

    if (dialog.exec() == QDialog::Accepted)
    {
        ...
    }
}
```

dialog 在作用范围结束时被删除

构造规范

- 几乎所有的 **QObject** 都有一个默认为空值的父对象。

```
QObject(QObject *parent=0);
```

- **QWidget** 的父对象是其它 **QWidget**类
- 为了方便倾向于提供多种构造（包括只带有父对象的一种）

```
QPushButton(QWidget *parent=0);
QPushButton(const QString &text, QWidget *parent=0);
QPushButton(const QIcon &icon, const QString &text, QWidget *parent=0);
```

- 父对象通常是带缺省值的第一个参数。

```
QLabel(const QString &text, QWidget *parent=0, Qt::WindowFlags f=0);
```

信号/槽机制

回调函数

- 回调函数是一个通过函数指针调用的函数。
- 如果把函数的指针(地址)作为参数传递给另一个函数，当这个指针被用来调用它所指向的函数时，我们就说这是回调函数。
- 回调函数不是由该函数的实现方式直接调用，而是在特定的事件或条件发生时由另外的一方调用的，用于对该事件或条件进行响应。

Signal/Slot机制

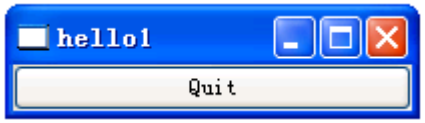
- Qt程序中，事件处理的方式也是回调，但与回调函数所不同的是，事件的发出和接收采用了信号（signal）和插槽（slot）机制，无须调用翻译表，是类型安全的回调。
- 类似于观察者设计模式
  - 信号槽机制可以在对象之间彼此并不了解的情况下将它们的行为联系起来。
  - 槽函数能和信号相连接，只要信号发出了，这个槽函数就会自动被调用。
- 利用信号和插槽进行对象间的通信是Qt的最主要特征之一。

Signal/Slot机制（续）

- 当对象状态发生改变的时候，发出signal通知所有的slot接收signal，尽管它并不知道哪些函数定义了slot，而slot也同样不知道要接收怎样的signal
- signal和slot机制真正实现了封装的概念，slot除了接收signal之外和其它的成员函数没有什么不同
- signal和slot之间是多对多的对应关系。
- 在QObject中实现：也就是说，要使任何类支持信号/槽机制，必须继承QObject类

Signal/Slot实例

```
#include <QApplication>
#include <QPushButton>
int main (int argc, char *argv [])
{
    QApplication app (argc, argv);
    QPushButton *button = new QPushButton (“Qui”,0);
    QObject::connect (button, SIGNAL (clicked ()), &app, SLOT (quit ()));
    button->show ();
    return app. exec ();
}
```



- **Qt程序的窗口部件发射信号（signals）来指出一个用户的动作或者是状态的变化。**
- 当信号被发射的时候，和信号相连的槽就会自动执行。
- “信号和槽”机制用于**Qt**对象间的通讯。

Signal和Slot的声明

- 在Qt程序设计中，凡是包含signal和slot的类中都要加上Q_OBJECT宏定义
- 信号是一个类的成员方法，该方法的实现是由meta-object自动实现的
  - 对于开发者只需要在类中声明这个信号，并不需要实现。
- 下面的例子给出了如何在一个类中定义signal和slot：

```
class Student : public QObject
{
        Q_OBJECT
public:
        Student() { myMark = 0; }
        int mark() const { return myMark; }
public slots:
        void setMark(int newMark);
signals:
        void markChanged(int newMark);
private:
        int myMark;
};
```

Signal和Slot的声明（续）

- signal的发出一般在事件的处理函数中，利用emit发出signal
- 在下面的例子中在在事件处理结束后发出signal

```
void Student::setMark(int newMark)
{
        if (newMark!= myMark) {
          myMark = newMark;
          emit markChanged(myMark);
        }
}
```

Signal和Slot的声明（续）

- 槽（slot）和普通的c++成员函数很像。
  - 槽是类的一个成员方法，当信号触发时该方法执行。
  - 可以是虚函数（virtual）、可被重载（overload）、可以是公有的（public）、保护的（protective）或者私有的（private）。
  - 可以象任何c++成员函数一样被直接调用，可以传递任何类型的参数，可以使用默认参数。
- 槽不同于信号，需要开发者自己去实现。

什么是槽?

- 槽在各种槽段（section）中定义。

```
public slots:
    void aPublicSlot();
protected slots:
    void aProtectedSlot();
private slots:
    void aPrivateSlot();
```

- 槽可以有返回值。
- 任何数量的信号都可以关联到同一个槽。

```
connect(src, SIGNAL(sig()), dest, SLOT(slt()));
```

- 槽以一个普通的函数来实现。
- 槽可以作为普通函数被调用。

什么是信号?

- 信号在信号段（**section**）中定义

```
signals:
   void aSignal();
```

- 信号总是返回空
- 信号总是不必实现函数体
  - 由**moc**来提供实现
- 信号可以关联到任意数量的槽上
- 通常产生一个直接调用，但可以在线程之间作为事件来传递，甚至可以用在套接字之间(使用第三方类)
- 槽能以任意次序被激发
- 信号使用**emit** 关键字发射出去

```
emit aSignal();
```

Signal和Slot的连接

- 为了能够接受到信号，信号和槽需要使用connect()函数关联起来。
- connect()函数是QObject类的成员函数，它能够连接signal和slot，也可以用来连接signal和signal
- 函数原形如下：

```
bool QObject::connect(sender, SIGNAL(valueChanged(int)),
                      receiver, SLOT(display(int)));
```

- sender和receiver是QObject对象指针。
- SIGNALE()和SLOT()宏的作用是把他们的参数转换成字符串。
- 其他重载的connect函数形式，请参考QT帮助文档

Signal和Slot的连接（续）

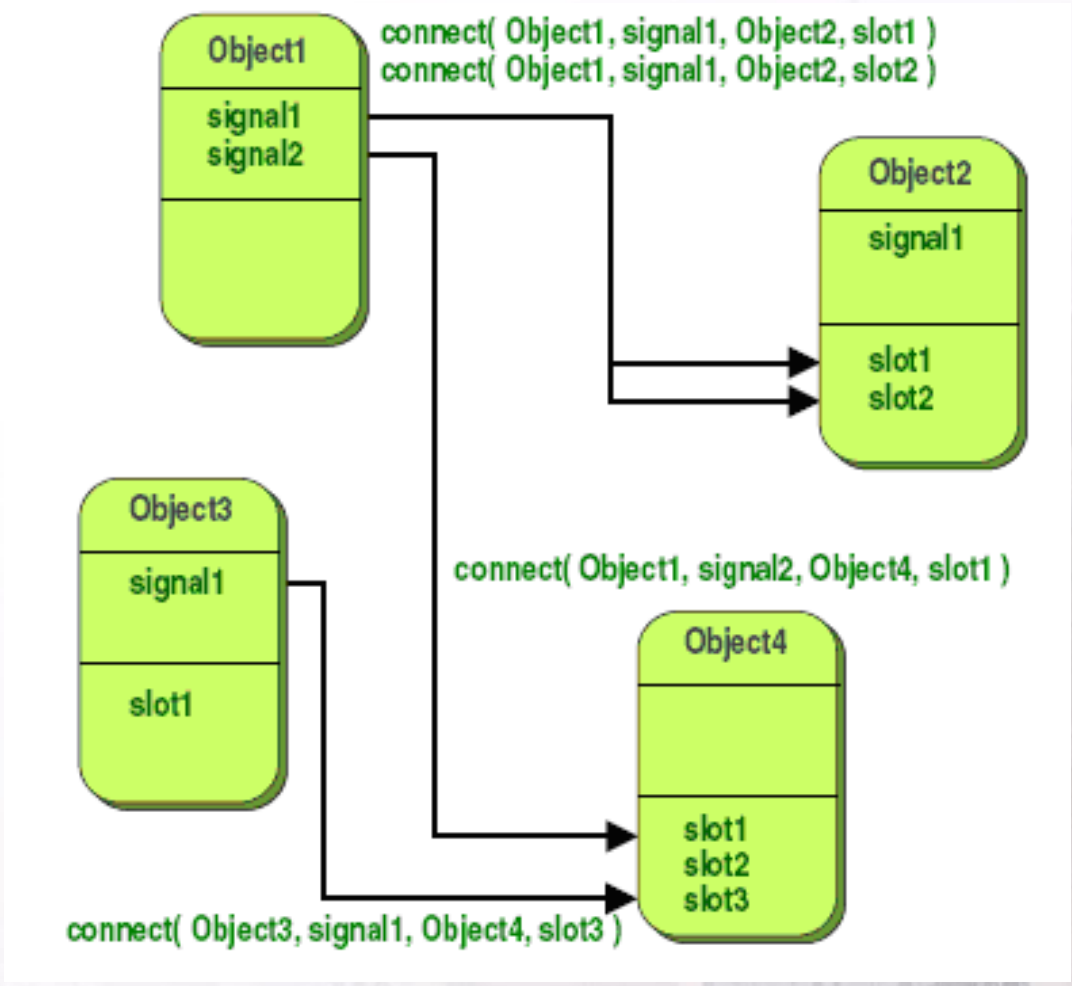
- 连接规则
  - 一个信号可以连接到多个槽
  - 多个信号可以连接到同一个槽
  - 一个信号可以和另一个信号相连
  - 连接可以被删除

Signal和Slot的连接（续）

- connect()函数举例：

```
QLabel *label = new QLabel;
QScrollBar *scroll = new QScrollBar;
QObject::connect( scroll,
                  SIGNAL(valueChanged(int)),
                  label,
                  SLOT(setNum(int)) );
```

Signal和Slot的连接（续）



Signal和Slot的连接（续）

- 同一个信号连接多个插槽

```
connect(slider, SIGNAL(valueChanged(int)),
        spinBox, SLOT(setValue(int)));
connect(slider, SIGNAL(valueChanged(int)),
        this, SLOT(updateStatusBarIndicator(int)));
```

- 多个信号连接到同一个插槽

```
connect(lcd, SIGNAL(overflow()),
        this, SLOT(handleMathError()));
connect(calculator, SIGNAL(divisionByZero()),
        this, SLOT(handleMathError()));
```

Signal和Slot的连接（续）

- 一个信号连接到另一个信号

```
connect(lineEdit, SIGNAL(textChanged(const QString &)),
        this, SIGNAL(updateRecord(const QString &)));
```

- 取消一个连接

```
disconnect(lcd,SIGNAL(overflow()),
           this, SLOT(handleMathError()));
```

  - 取消一个连接不是很常用，因为Qt会在一个对象被删除后自动取消这个对象所包含的所有的连接

信号与槽机制深入

- 信号发生后
  - 如果信号和槽实现在同一个线程中，当信号产生的时候，与它关联的槽就会马上得到执行
  - 如果信号和槽不在同一个线程中，槽的执行可能会有延迟(next event loop)
- 相关联的信号和槽必须满足一定条件
  - 信号的参数可以多于槽的参数,多余的参数被忽略，反之则不行
  - 信号和槽函数必须有着相同的参数类型及顺序
  - 没有编译时的错误检查，只有运行期检查
    - 如果参数类型不匹配，或者信号和槽不存在，应用程序在debug状态下时，Qt会在运行期间给出警告。
    - 如果信号和槽连接时包含了参数的名字，Qt也将会给出警告。

信号与槽机制深入（续）

- signal和slot只是对于回调函数一个比较安全的封装（wrapper）
  - slot对应回调函数，signal则相当于触发回调函数的方法。
  - 但QApplication可以模拟异步的方式。
- 如果程序只是简单使用Qt的基本类，或者从QtObject派生而来的自定义类，而不是QApplication的话，肯定不是异步机制
  - 实验：在emit之后printf(“emit\n”);在slot中printf(“slot\n”)，结果是首先打印slot然后emit，这就表明了emit调用陷入了slot中。
  - qApp在事件循环处理中截取所有emit的signal，然后调用相应的slots，就像回调函数一样

建立关联

QObject*

QObject::connect(src, SIGNAL(signature), dest, SLOT(signature));

<function name> (<arg type>...)

签名由函数名和参数类型组成。不允许有变量名或值。

setTitle(QString text)
setValue(42)

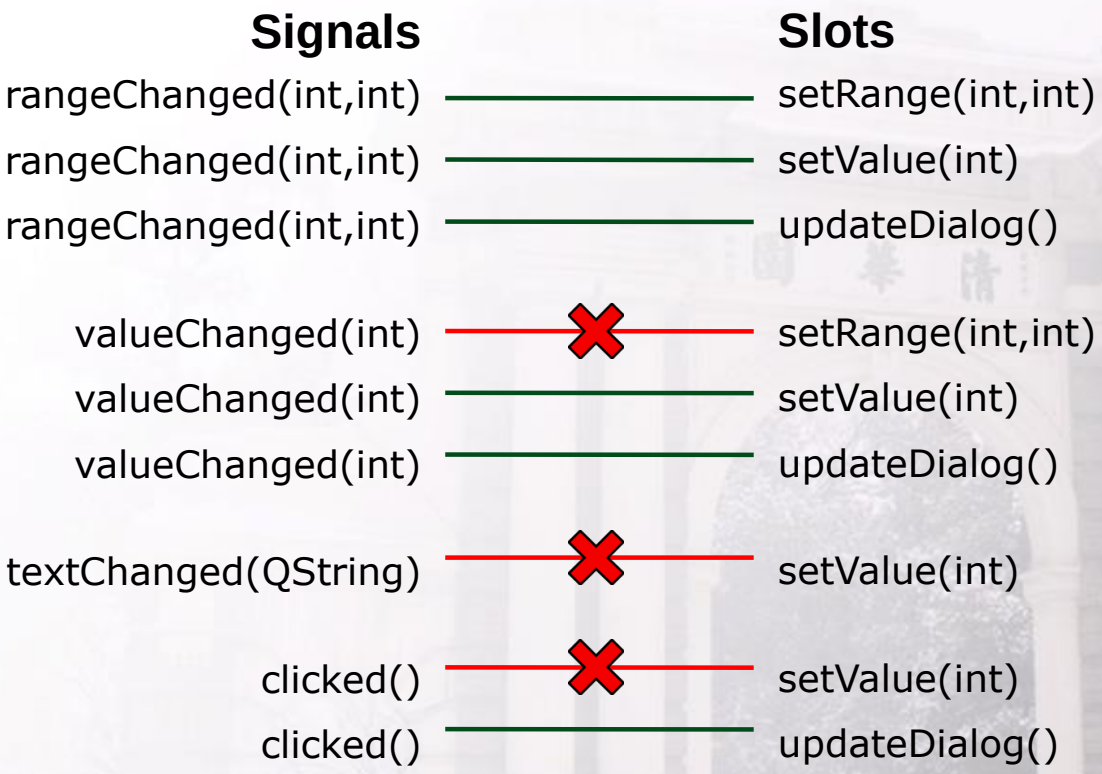
setItem(ItemClass)

自定义类型降低了可重用性

clicked()
toggled(bool)
setText(QString)
textChanged(QString)
rangeChanged(int,int)

建立关联

- Qt 参数可以忽略，但不能无中生有。



自动关联

- 使用**Qt Designer**，它很便捷地在接口和用户代码之间提供自动关联。

on_ *object name* _ *signal name* (*signal parameters*)

on_addButton_clicked();

on_deleteButton_clicked();

on_listWidget_currentItemChanged(QListWidgetItem*,QListWidgetItem*)

- 通过调用**QMetaObject::connectSlotsByName**触发
- 当命名时考虑重用性
  - 比较 **on_widget_signal** 和 **updatePageMargins**

updatePageMargins可以关联到一定数量信号或直接调用。

Signal/Slot链接举例

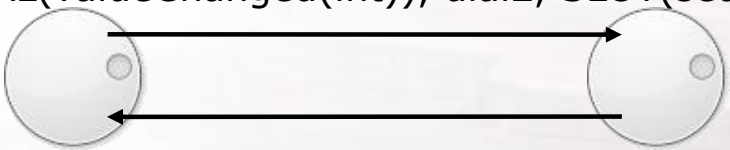
```
int main(int argc, char *argv[])
{
   QApplication app(argc, argv);

   QPushButton *button = new QPushButton("Quit");
   QObject::connect(button, SIGNAL(clicked()),
                    &app, SLOT(quit()));
   button->show();

   return app.exec();
}
```

Qt程序的窗口部件发射信号（signals）来指出一个用户的动作或者是状态的变化。当信号被发射的时候，和信号相连的槽就会自动执行。“信号和槽”机制用于Qt对象间的通讯。

值同步

- 双向连接

```
connect(dial1, SIGNAL(valueChanged(int)), dial2, SLOT(setValue(int)));
```



```
connect(dial2, SIGNAL(valueChanged(int)), dial1, SLOT(setValue(int)));
```

- 无限循环必须停止——没有信号被发射，除非发生实际的变化。

```
void QDial::setValue(int v)
{
    if(v==m_value)
        return;
    ...
```

这就是负责发射信号的所有代码—在您自己的类中不要忘记它。

自定义信号和槽

在这里添加一个通知信号。

```
class AngleObject : public QObject
{
    Q_OBJECT
    Q_PROPERTY(qreal angle READ angle WRITE setAngle NOTIFY angleChanged)

public:
    AngleObject(qreal angle, QObject *parent = 0);
    qreal angle() const;

public slots:
    void setAngle(qreal);

signals:
    void angleChanged(qreal);

private:
    qreal m_angle;
};
```

setter构造自然槽。

信号匹配setter

setter实现细节

```
void AngleObject::setAngle(qreal angle)
{
    if(m_angle == angle)
        return;

    m_angle = angle;
    emit angleChanged(m_angle);
}
```

防止无限循环。
不要忘记!

更新内部状态，然后发射信号。

信号是被“保护”的，他们可以从派生类发射。

与值关联?

- 一种常见情况是，希望在关联声明中传递一个值。

```
connect(key, SIGNAL(clicked()), this, SLOT(keyPressed(1)));
```

- 例如, 键盘实例

- 这不是有效的 --它将不会关联。

与值关联?

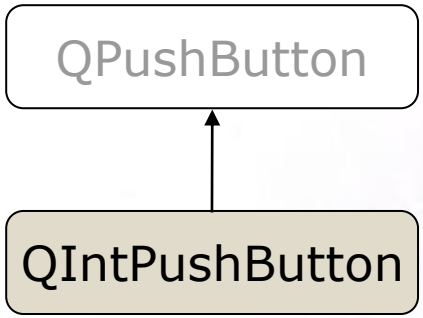
- 解决方法 #1: 多个槽

connections

```
{
    ...

public slots:
    void key1Pressed();
    void key2Pressed();
    void key3Pressed();
    void key4Pressed();
    void key5Pressed();
    void key6Pressed();
    void key7Pressed();
    void key8Pressed();
    void key9Pressed();
    void key0Pressed();

    ...
}
```

与值关联?

- 解决方法 #2: 子类发射器和增加信号




```
{
    ...

signals:
    void clicked(int);

    ...
}
```

```
{
    QIntPushButton *b;

    b=new QIntPushButton(1);
    connect(b, SIGNAL(clicked(int)),
        this, SLOT(keyPressed(int)));

    b=new QIntPushButton(2);
    connect(b, SIGNAL(clicked(int)),
        this, SLOT(keyPressed(int)));

    b=new QIntPushButton(3);
    connect(b, SIGNAL(clicked(int)),
        this, SLOT(keyPressed(int)));

    ...
}
```

解决方案评价

- #1: 多个槽
  - 许多槽包含几乎相同的代码
  - 难于维护 (一个小的变化影响所有槽)
  - 难于扩展 (每次都要新建槽)
- #2:子类发射器和增加信号
  - 额外的专用类 (难于重用)
  - 难于扩展 (每个情况需新建子类)

信号映射器

- **QSignalMapper 类解决了这个问题**
  - 将每个值映射到每个发射器
  - 介于可重用类之间

```
{
    QSignalMapper *m = QSignalMapper(this);
    QPushButton *b;

    b=new QPushButton("1");
    connect(b, SIGNAL(clicked()),
            m, SLOT(map()));
    m->setMapping(b, 1);

    ...

    connect(m, SIGNAL(mapped(int)), this, SLOT(keyPressed(int)));
}
```

创建一个信号映射器

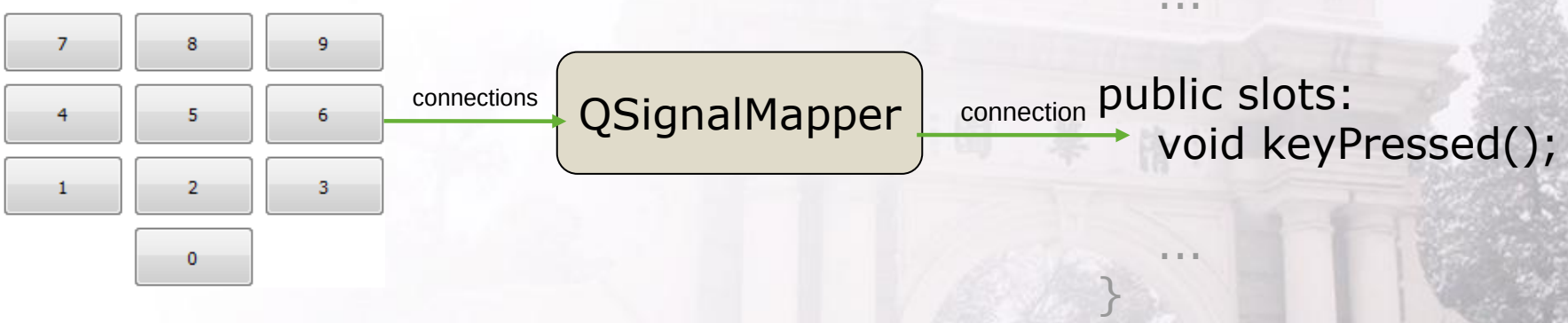
关联按钮到映射器

关联一个发射器和一个值。

关联映射器到槽上。

信号映射器

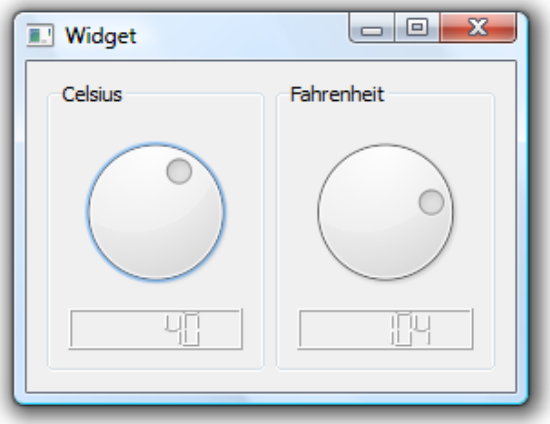
- 信号映射器把每一个按钮和值关联起来。这些值都被映射。



- 当一个值被映射，映射器发出携带关联的值的映射信号（**int**）。

简单实例：温度转换器

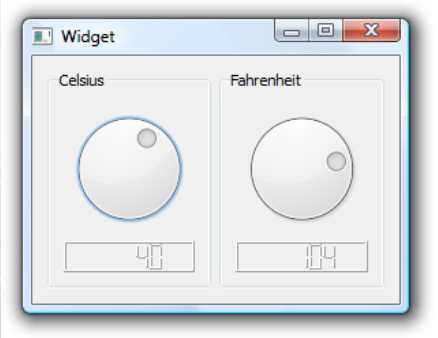
温度转换器



- 使用 **TempConverter** 类实现在摄氏与华氏之间的转换
- 当温度改变时发射信号。

温度转换器

- 对话窗口（dialog window）包含以下对象
  - 一个 **TempConverter** 实例
  - 两个 **QGroupBox** 部件（**widget**），每一个包含
    - 一个 **QDial** 部件
    - 一个 **QLCDNumber** 部件



温度转换器

```
class TempConverter : public QObject
{
    Q_OBJECT

public:
    TempConverter(int tempCelsius, QObject *parent = 0);

    int tempCelsius() const;
    int tempFahrenheit() const;

public slots:
    void setTempCelsius(int);
    void setTempFahrenheit(int);

signals:
    void tempCelsiusChanged(int);
    void tempFahrenheitChanged(int);

private:
    int m_tempCelsius;
};
```

QObject 作为父对象

先是Q_OBJECT 宏

父对象指针

读和写函数

当温度变化时发射信号。

在内部表示整数摄氏度。

温度转换器

- **setTempCelsius槽:**

```
void TempConverter::setTempCelsius(int tempCelsius)
{
    if(m_tempCelsius == tempCelsius)
        return;

    m_tempCelsius = tempCelsius;

    emit tempCelsiusChanged(m_tempCelsius);
    emit tempFahrenheitChanged(tempFahrenheit());
}
```

测试改变以中断递归

更新对象的状态

发射信号反映改变

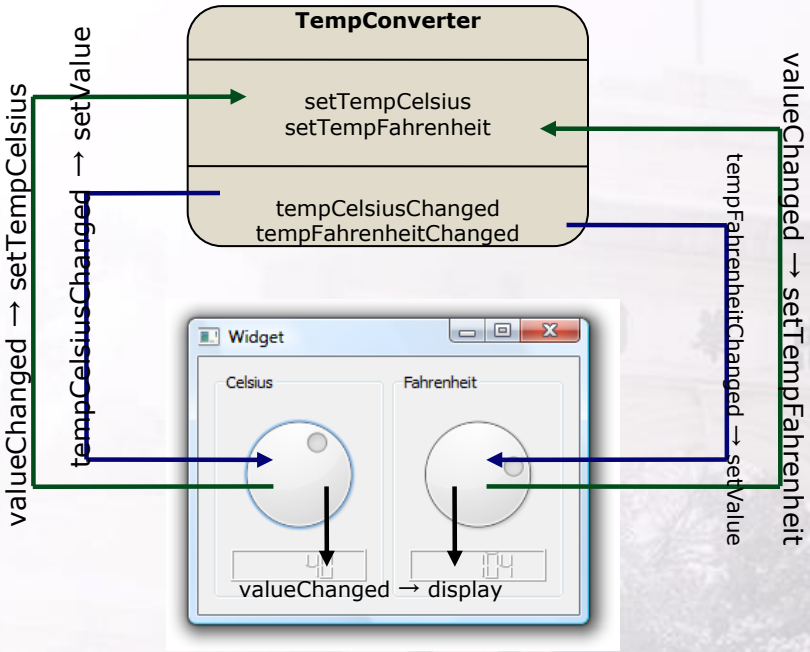
- **setTempFahrenheit槽:**

```
void TempConverter::setTempFahrenheit(int tempFahrenheit)
{
    int tempCelsius = (5.0/9.0)*(tempFahrenheit-32);
    setTempCelsius(tempCelsius);
}
```

转换，传递摄氏度是内部表现形式。

温度转换器

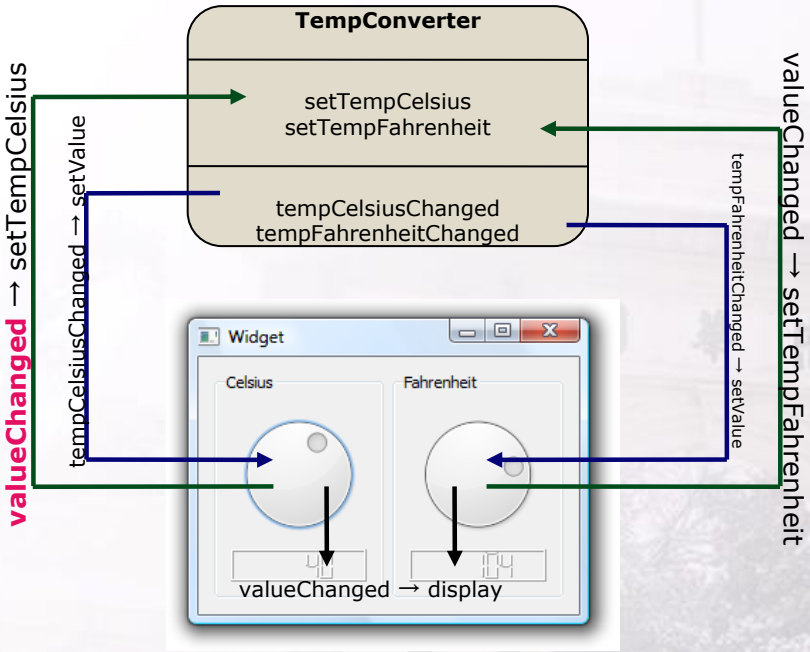
- 表盘通过 **TempConverter** 联系起来
- **LCD** 显示直接受表盘来驱动。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

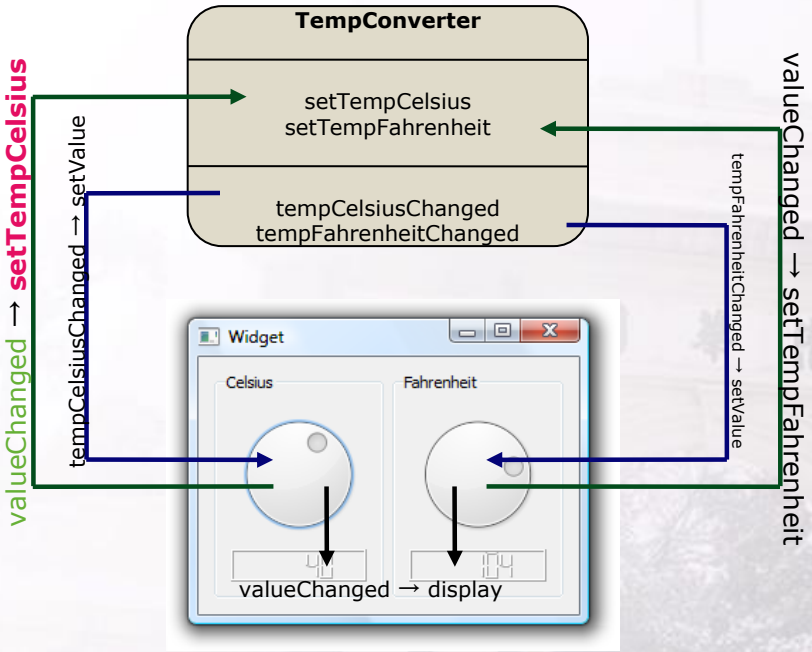
- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

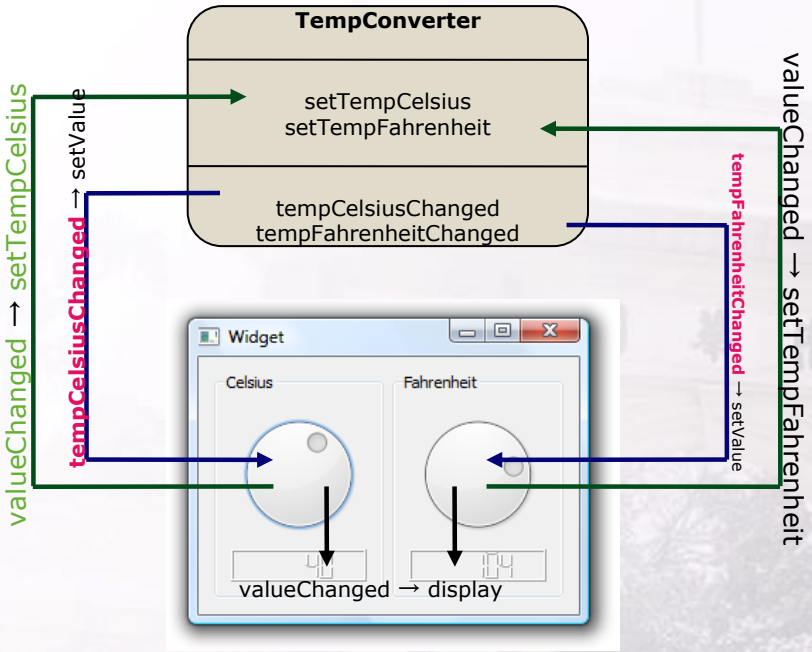
- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

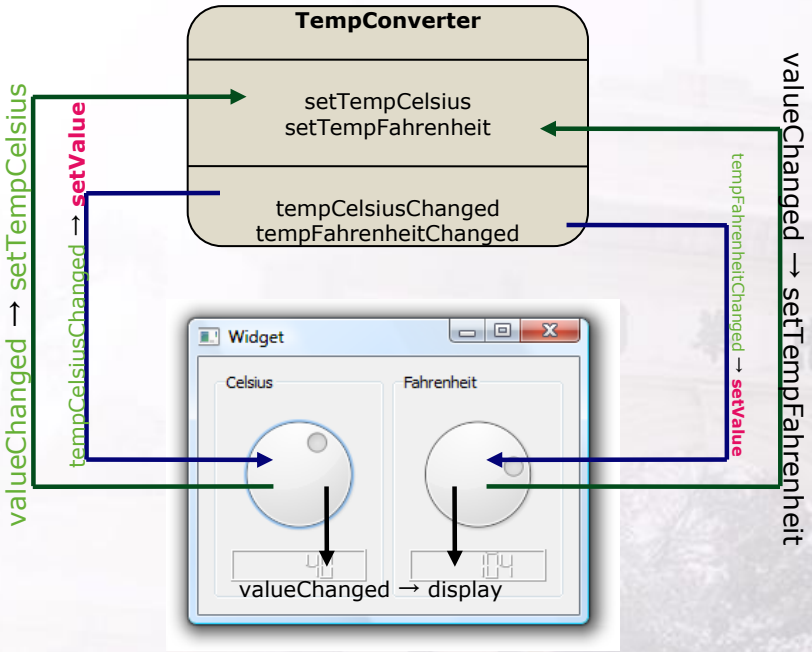
- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

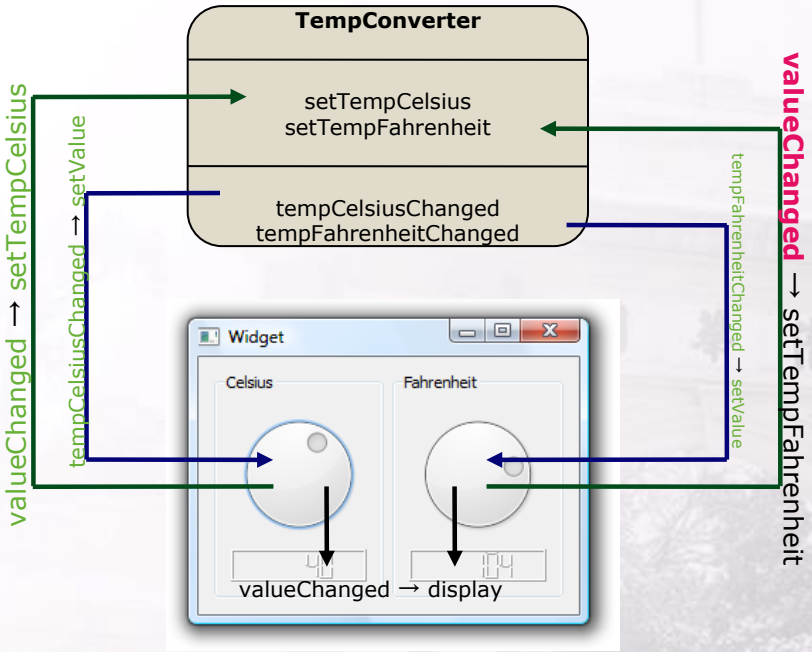
- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

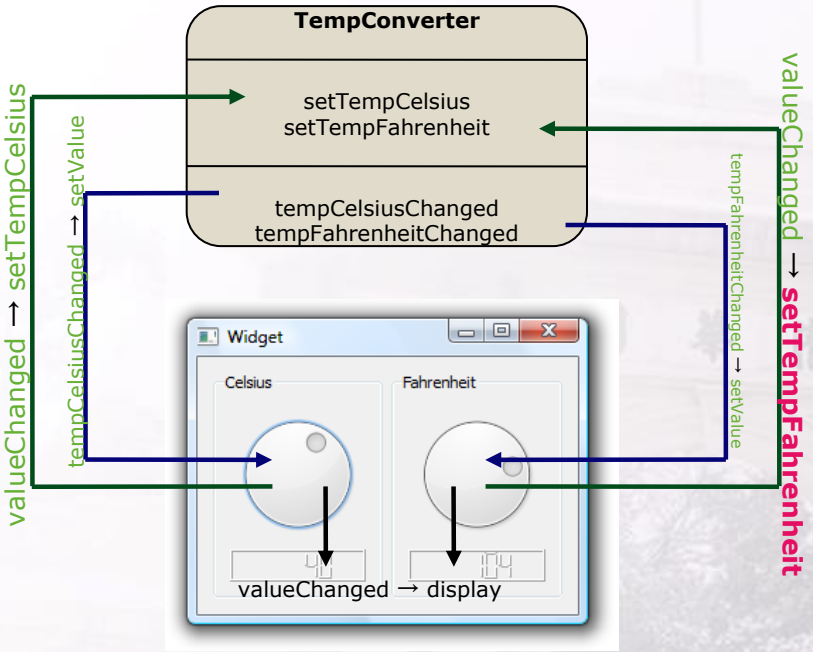
- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

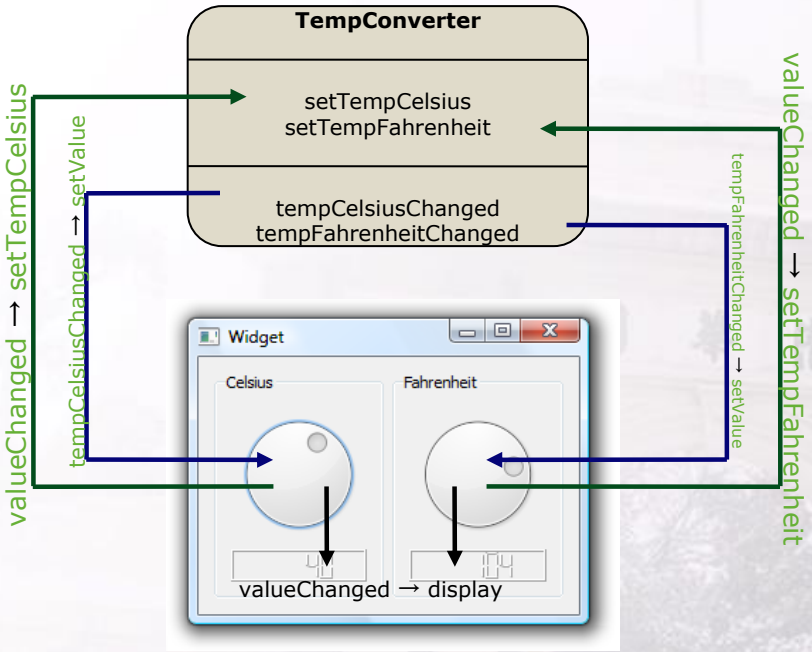
- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

温度转换器

- 用户调节摄氏度表盘。




```
connect(celsiusDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempCelsius(int)));
connect(celsiusDial, SIGNAL(valueChanged(int)), celsiusLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempCelsiusChanged(int)), celsiusDial, SLOT(setValue(int)));

connect(fahrenheitDial, SIGNAL(valueChanged(int)), tempConverter, SLOT(setTempFahrenheit(int)));
connect(fahrenheitDial, SIGNAL(valueChanged(int)), fahrenheitLcd, SLOT(display(int)));
connect(tempConverter, SIGNAL(tempFahrenheitChanged(int)), fahrenheitDial, SLOT(setValue(int)));
```

Qt的获取及参考资料

Qt工具下载及学习资源

- **Qt Creator官方下载**
  - 最新版本（建议Offline Installers）：qt-opensource-windows-x86-5.11.1.exe
  - http://download.qt.io/archive/qt/
  - “Configure Project”时选择MinGW编译器
- **Qt 帮助**
  - Qt Creator自带帮助文档
  - Qt在线帮助文档：http://doc.qt.io/qt-5/reference-overview.html
- **推荐学习Qt Examples And Tutorials**
  - http://doc.qt.io/qt-5/qtexamplesandtutorials.html （安装目录下找Examples文件夹）
- 《C++ Gui Qt4 编程》（官方教材，电子工业出版社）等书籍
- 百度、谷歌搜索

请自学Qt的发展历史

http://baike.baidu.com/view/23681.htm?fr=aladdin

谢谢！