

Qt编程

QT PROGRAMMING

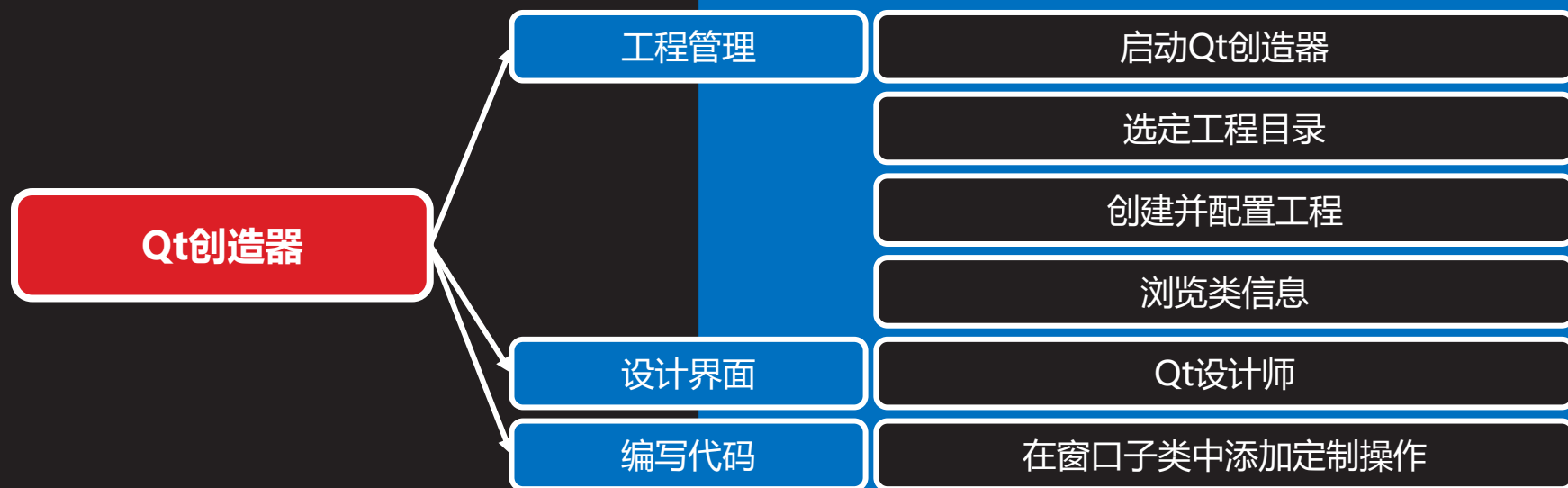
DAY03

内容

上午	09:00 ~ 09:30	作业讲解和回顾
	09:30 ~ 10:20	Qt创造器
	10:30 ~ 11:20	资源与图像
	11:30 ~ 12:20	目录与定时器
下午	14:00 ~ 14:50	鼠标与键盘
	15:00 ~ 15:50	贪食蛇游戏
	16:00 ~ 16:50	
	17:00 ~ 17:30	总结和答疑



Qt创造器

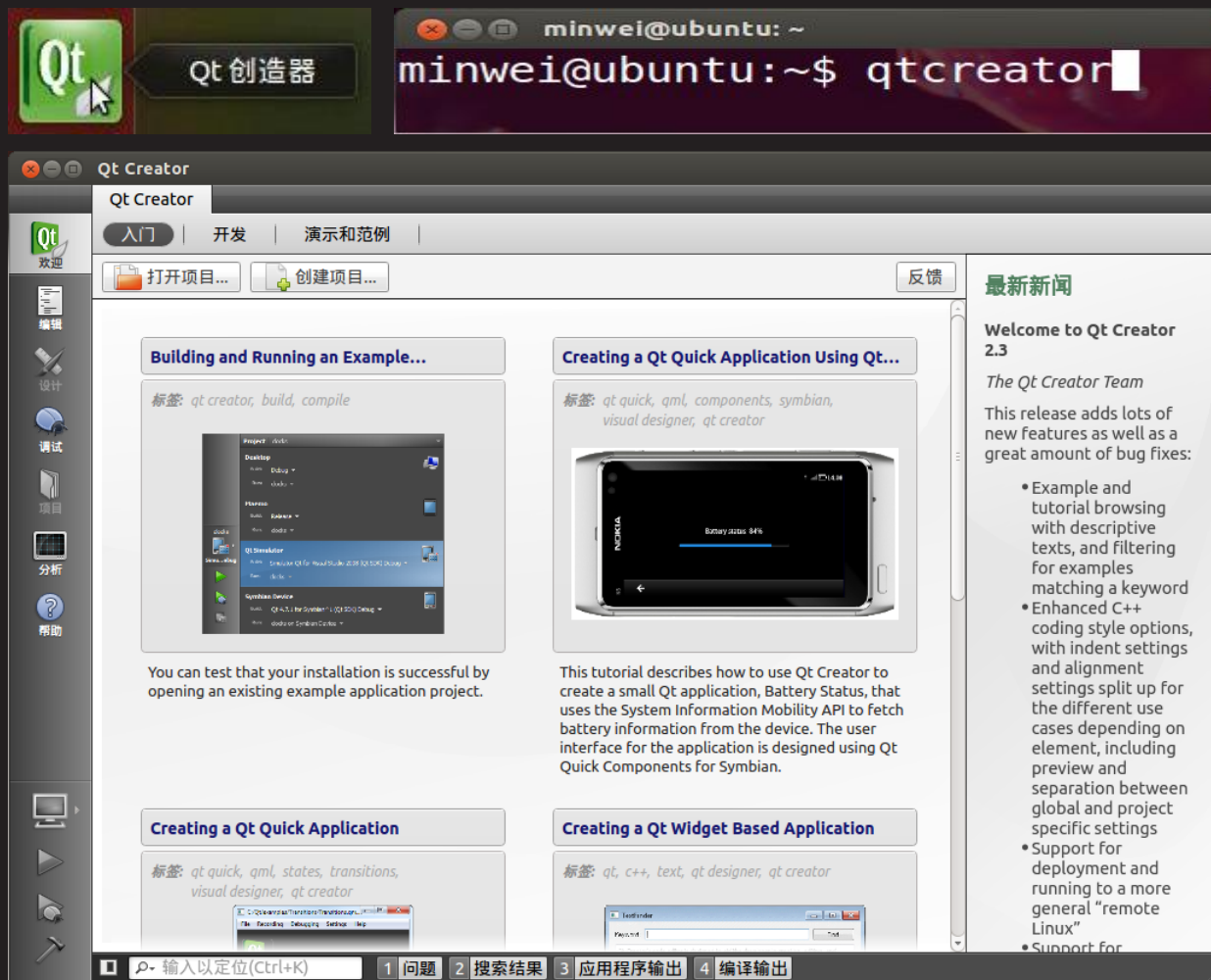


工程管理



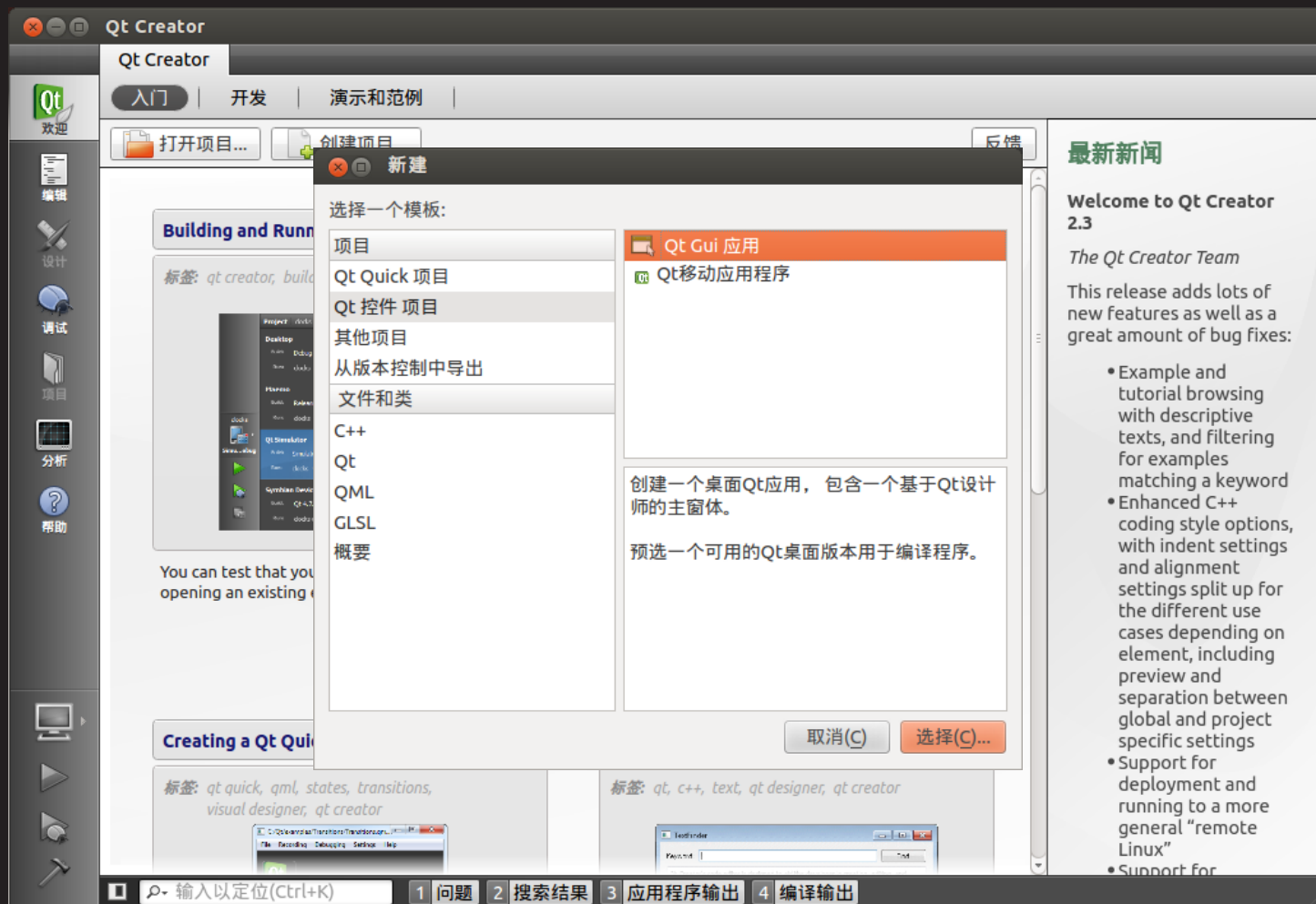
启动Qt创造器

- 无论是用鼠标点击桌面图标，还是从命令行终端输入 `qtcreator` 命令，都可以启动Qt创造器



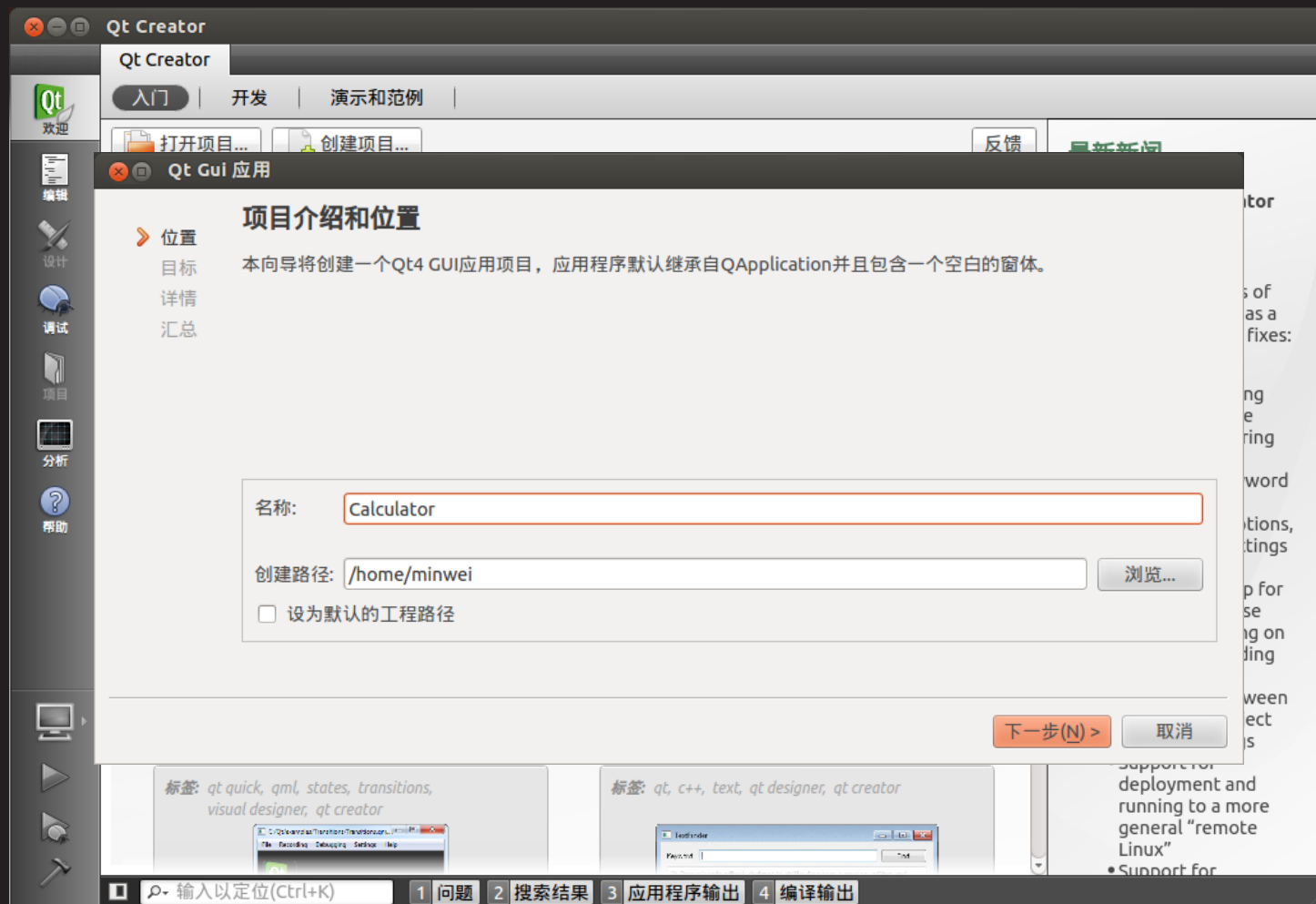
选定工程目录

- 选择菜单"文件/新建文件或工程..."，在"新建"对话框中依次选择"Qt控件项目"和"Qt Gui应用"，并点击"选择..."



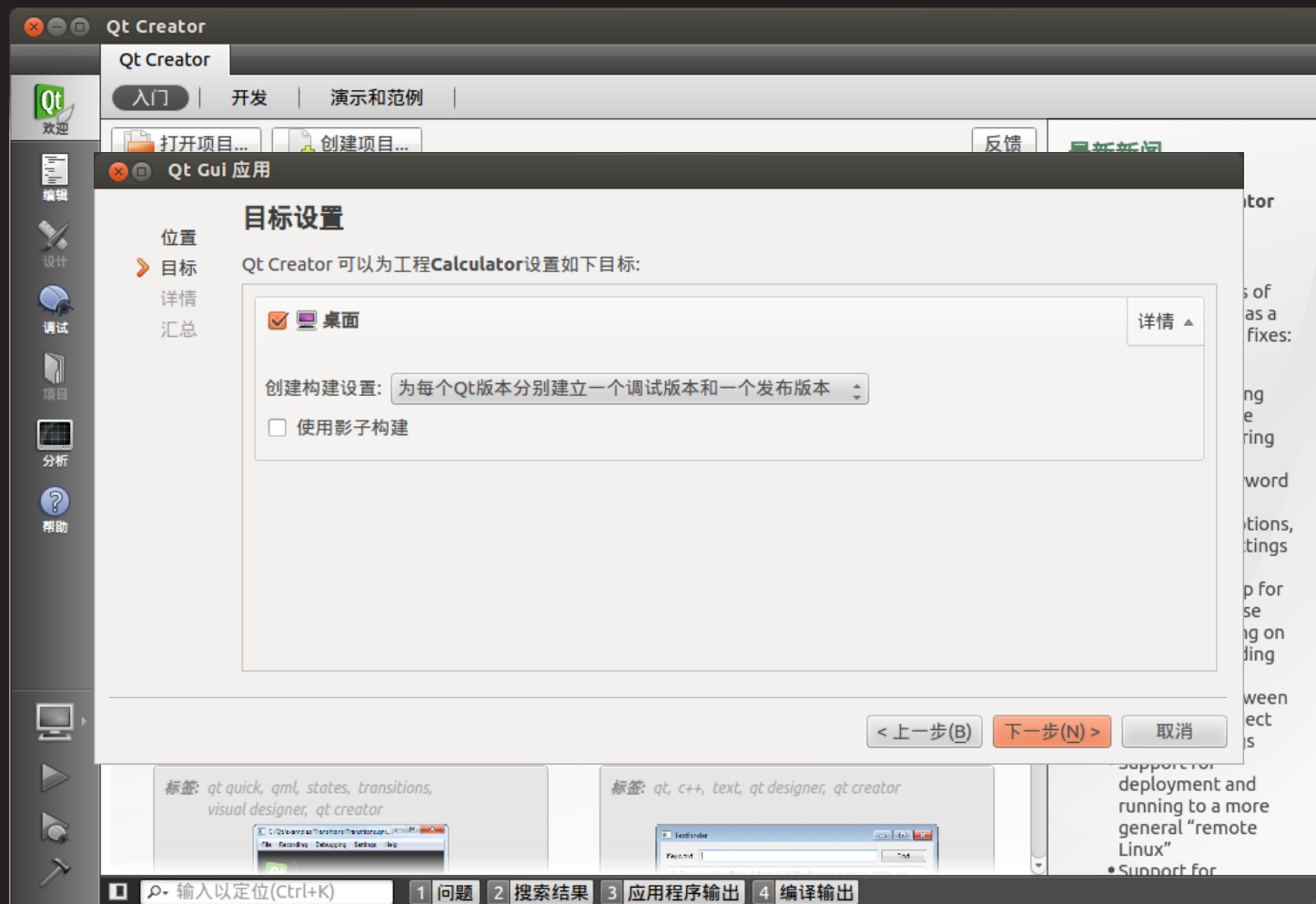
选定工程目录（续1）

- 在"项目介绍和位置"中指定项目的名称并选择存储路径，Qt会在该路径下创建工程目录，点击"下一步"



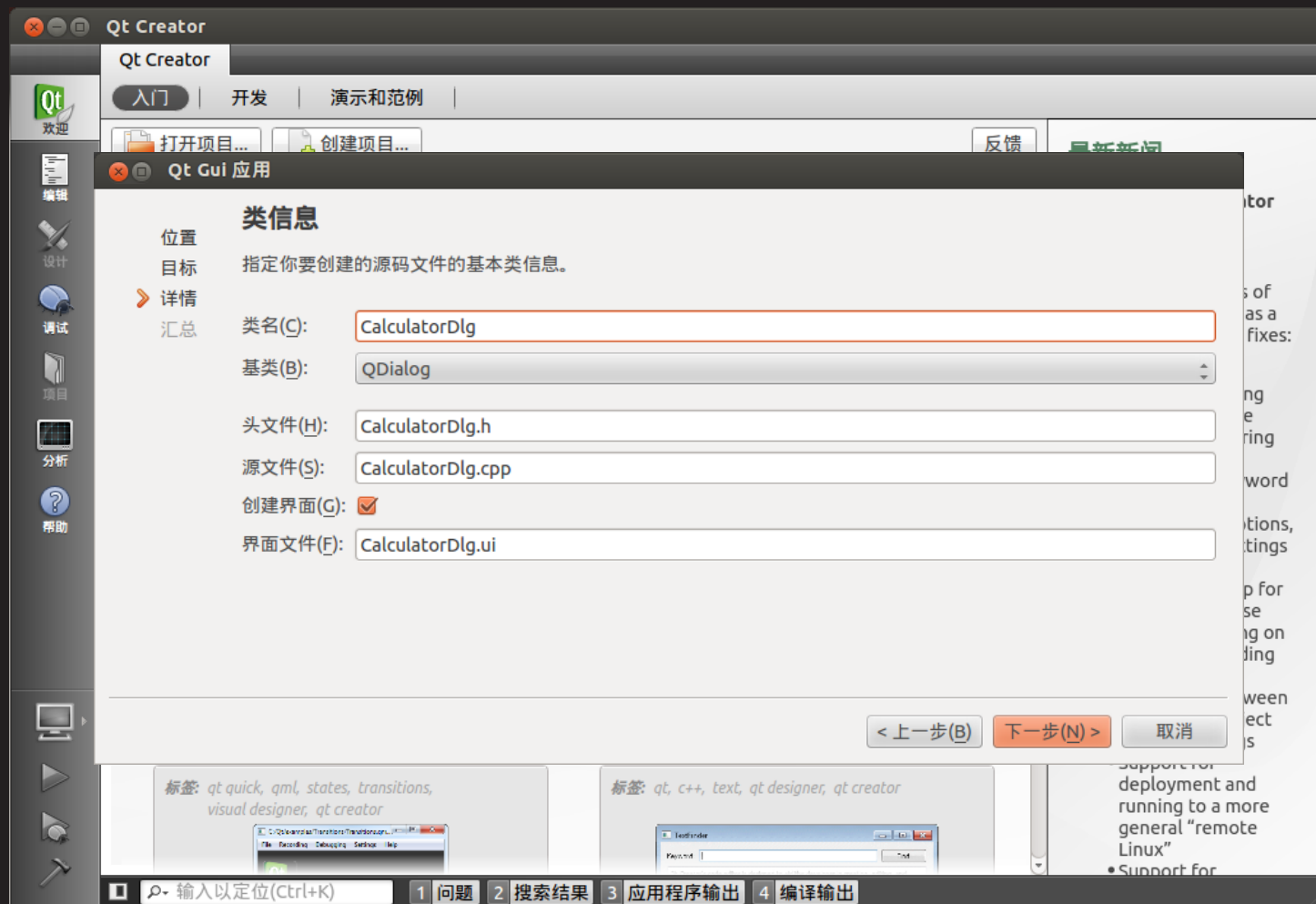
创建并配置工程

- 在“目标设置”中选择**桌面**，这通常也是缺省选项，如无特别需要，不要勾选“使用影子构建”，点击“下一步”



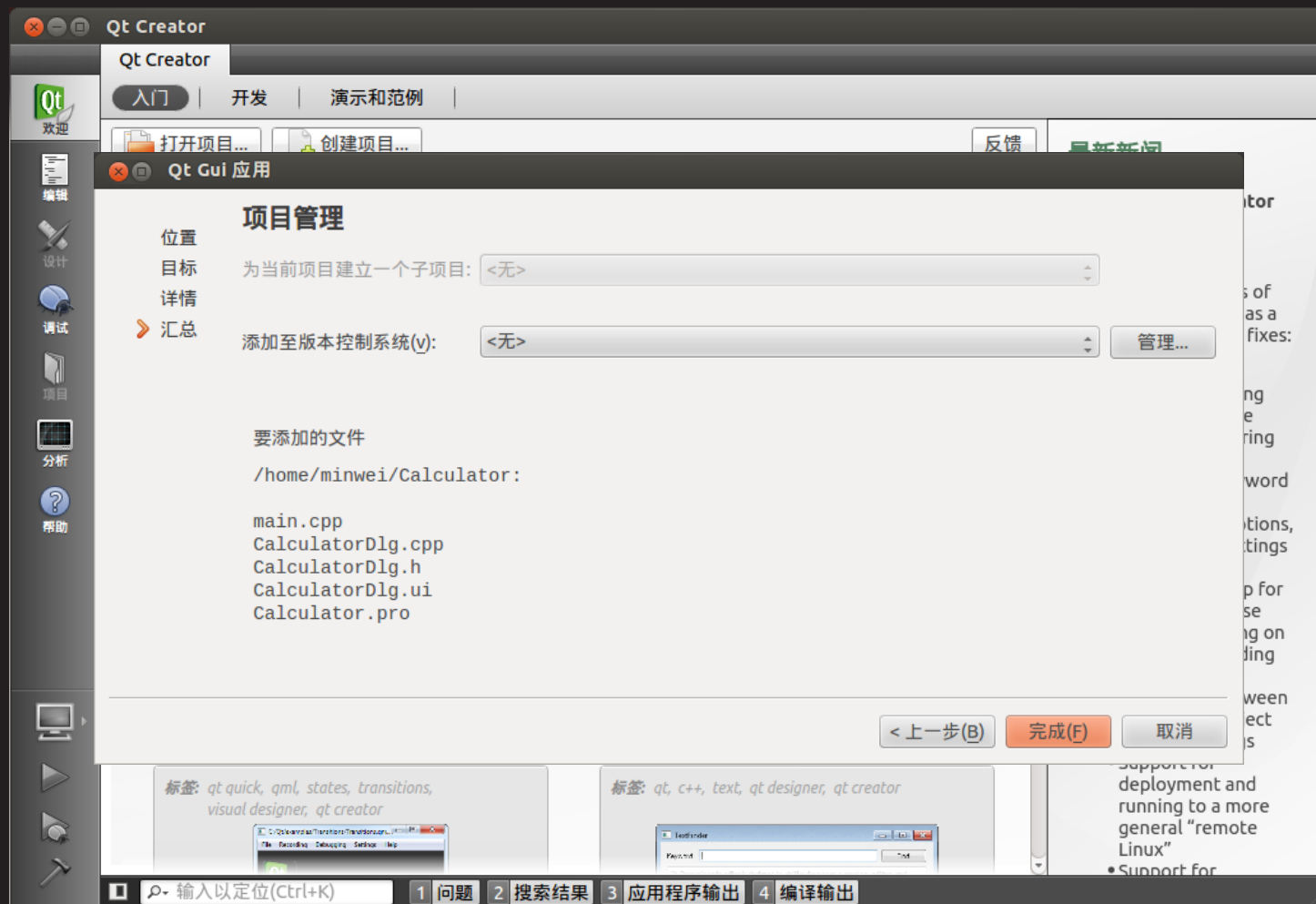
浏览类信息

- 在"类信息"中选择"**QDialog**"作为"基类"，并将"类名"设置为"**CalculatorDlg**"，勾选"**创建界面**"，点击"**下一步**"



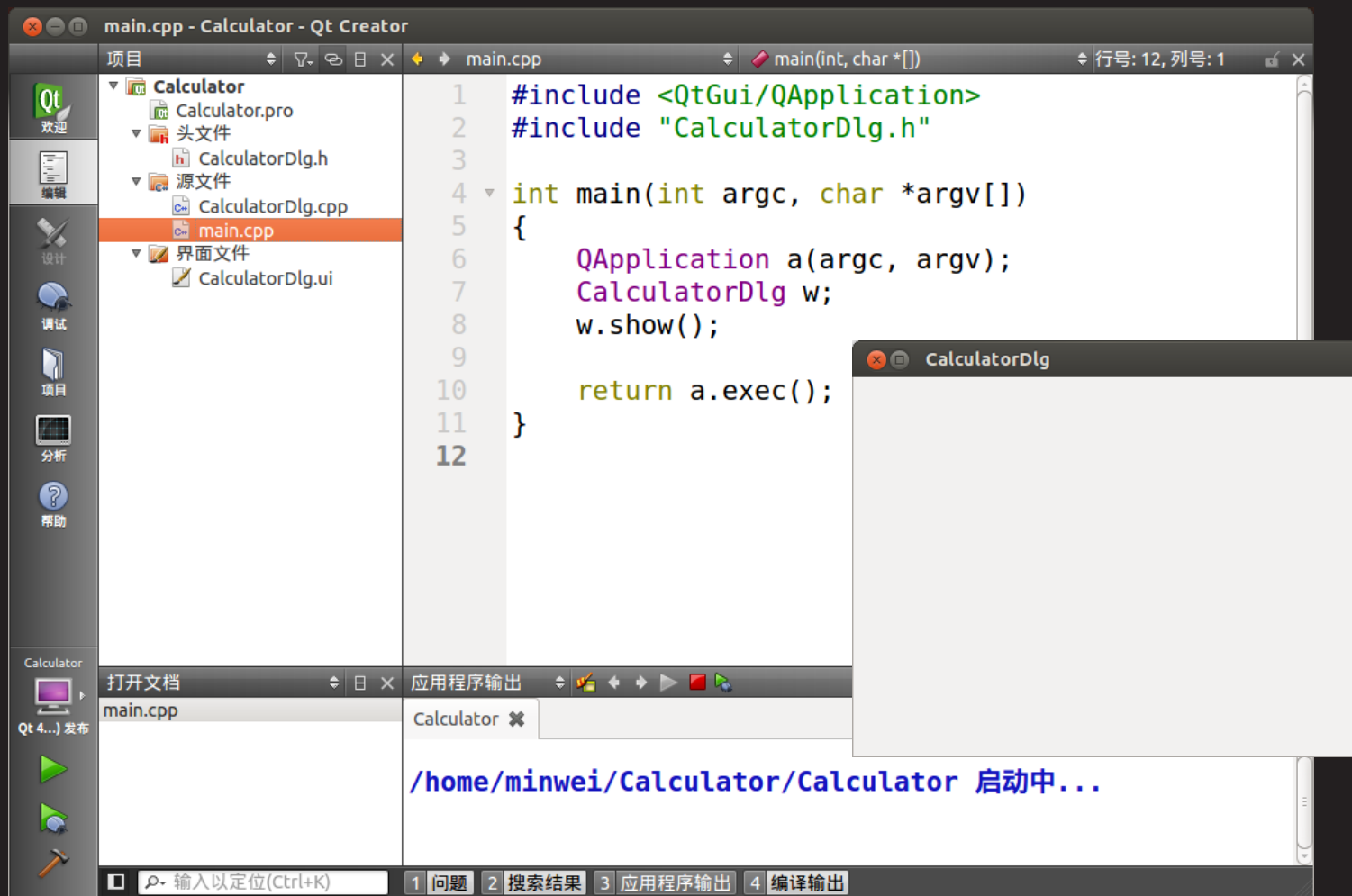
浏览类信息 (续1)

- 在"项目管理"中浏览项目摘要，配置**版本控制**(如果有的话)，点击"完成"。如果有问题可通过"上一步"返回修改



浏览类信息（续2）

- 截止目前虽然并没有编写一行代码，但是Qt创造器已经生成了基本的程序**框架**，并且是可编译、链接和运行的



浏览类信息（续3）

- Qt创造器所生成的基本程序框架包括
 - CalculatorDlg.ui
 - ✓ 界面描述文件，经uic编译生成ui_CalculatorDlg.h头文件，包含Ui::CalculatorDlg类的完整定义
 - CalculatorDlg.h
 - ✓ 声明CalculatorDlg类，继承自QDialog基类，通过成员变量ui组合一个Ui::CalculatorDlg对象
 - CalculatorDlg.cpp
 - ✓ 实现CalculatorDlg类，构造函数中通过Ui::CalculatorDlg类的setupUi()成员函数初始化界面
 - main.cpp
 - ✓ 定义main函数，创建并显示窗口
 - Calculator.pro和Makefile
 - ✓ 工程文件和构建脚本

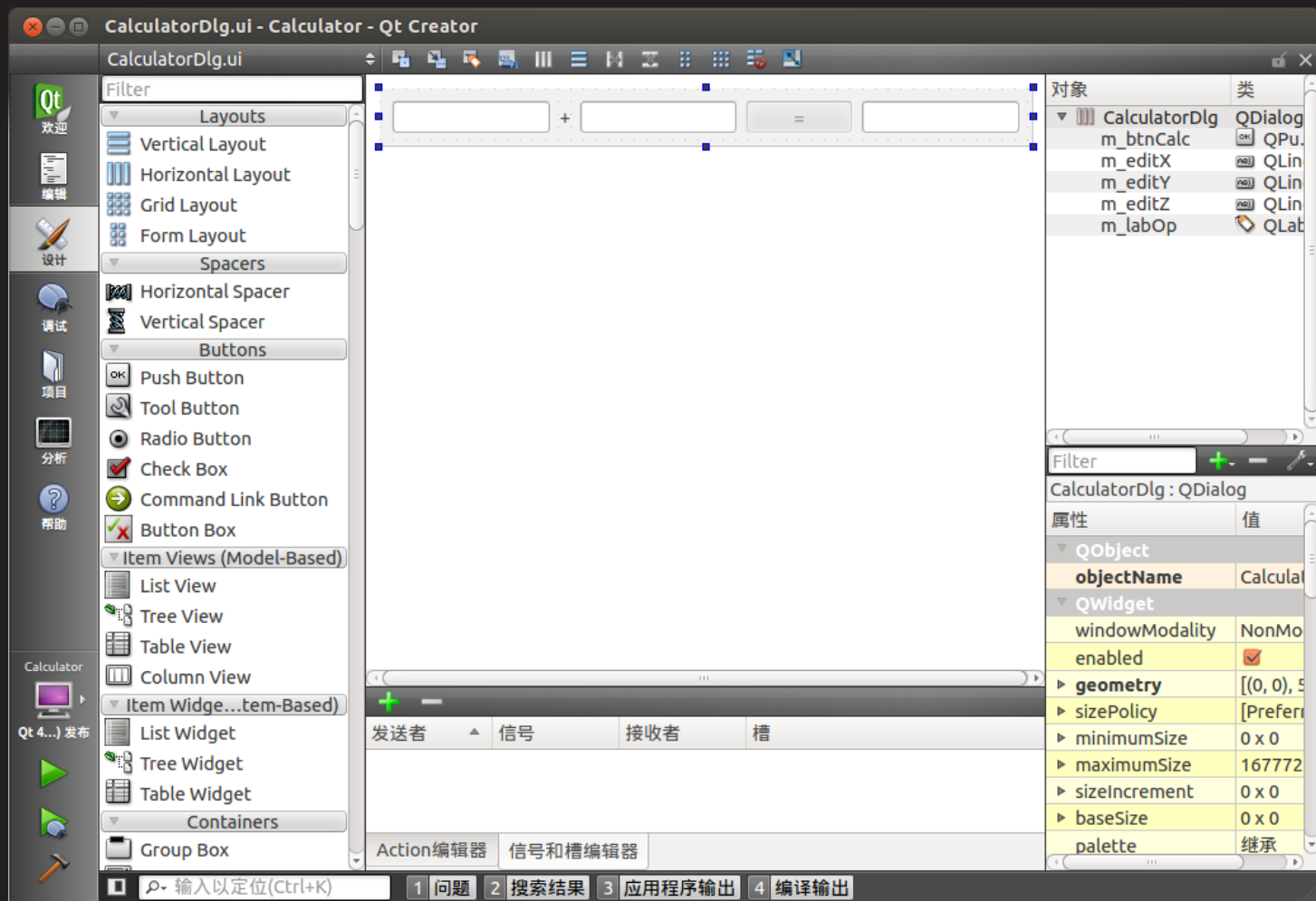


设计界面



Qt设计师

- 双击"项目"窗格中的"**CalculatorDlg.ui**"，程序会自动启动**Qt设计师**并打开界面文件，按照之前的方法设计界面

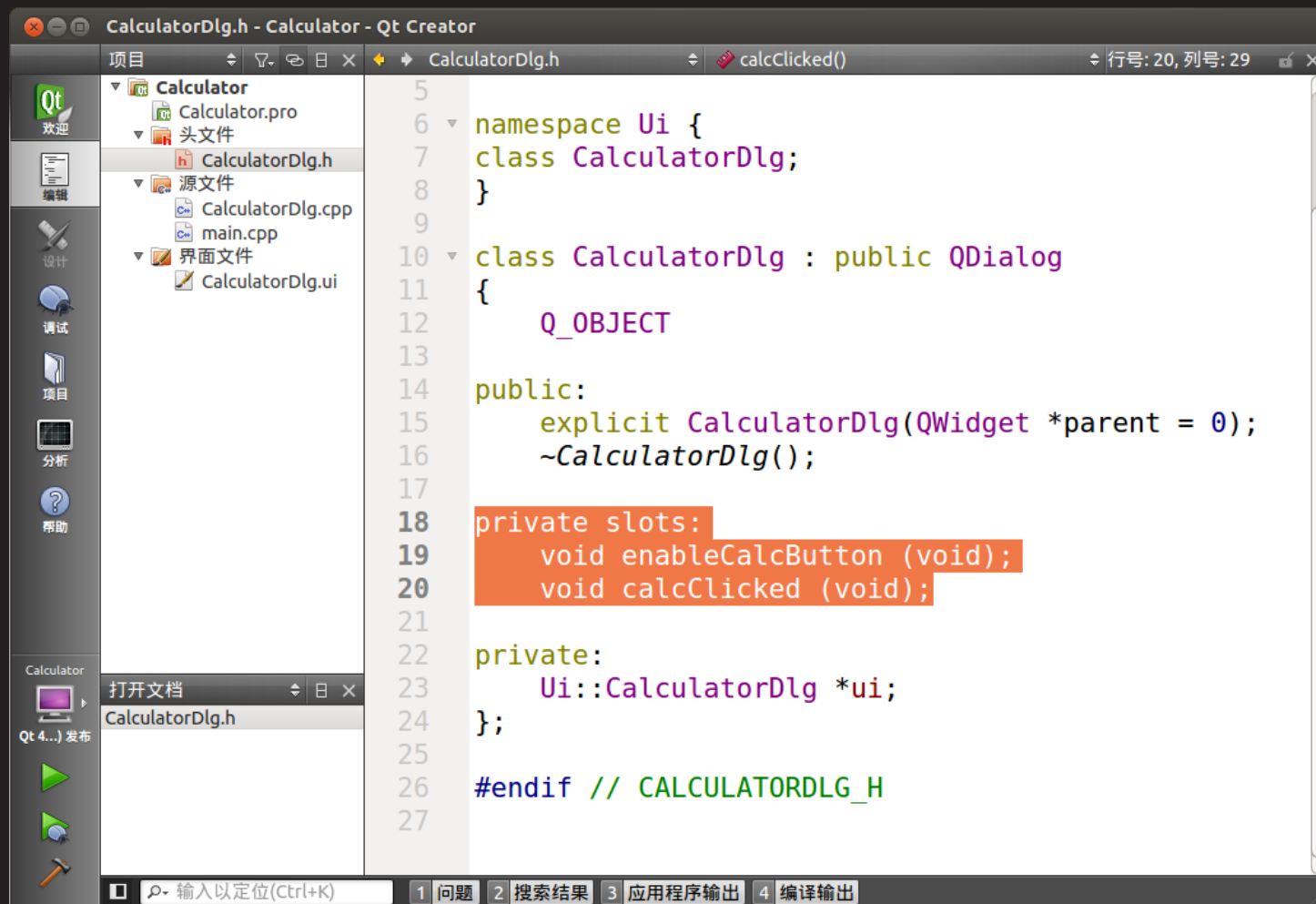


编写代码



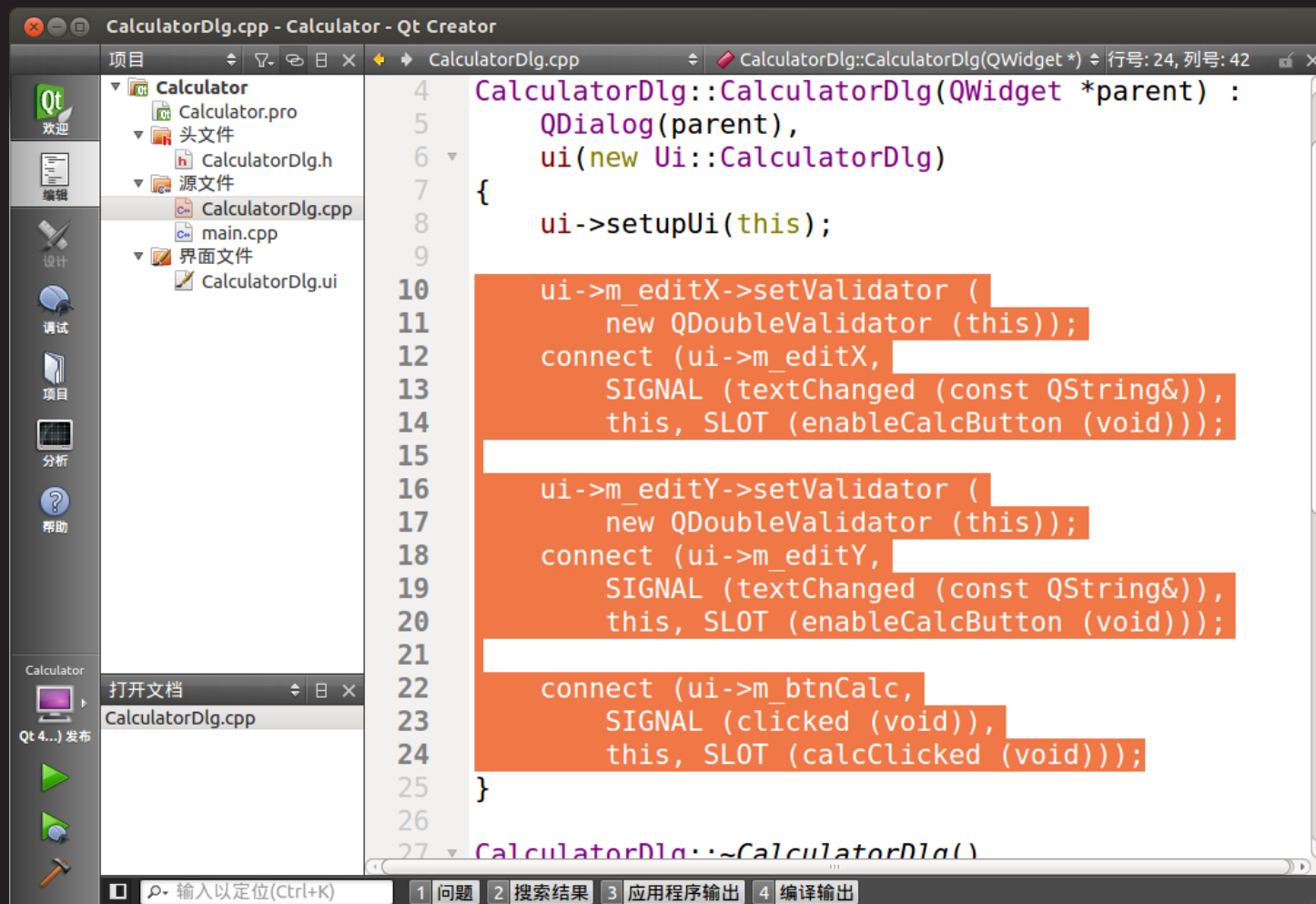
在窗口子类中添加定制操作

- 关闭CalculatorDlg.ui文件退出Qt设计师。双击"项目"窗格中的"CalculatorDlg.h"，加入两个槽函数的声明



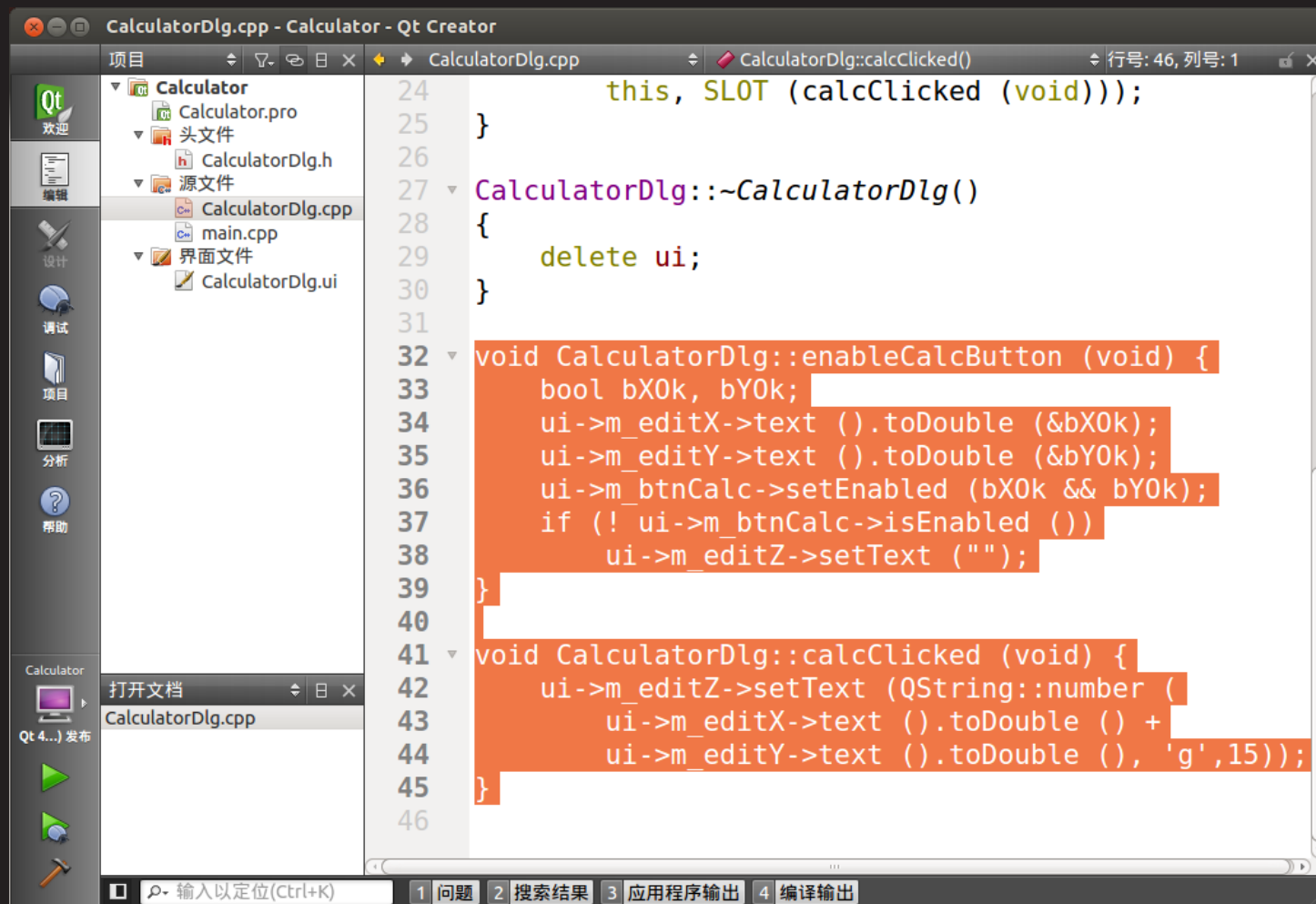
在窗口子类中添加定制操作（续1）

- 关闭CalcuaorDlg.h，双击"项目"窗格中的"CalculatorDlg.cpp"，在构造函数中添加初始化代码



在窗口子类中添加定制操作（续2）

- 继续编辑CalculatorDlg.cpp文件，加入两个槽函数enableCalcButton()和calcClicked()的定义



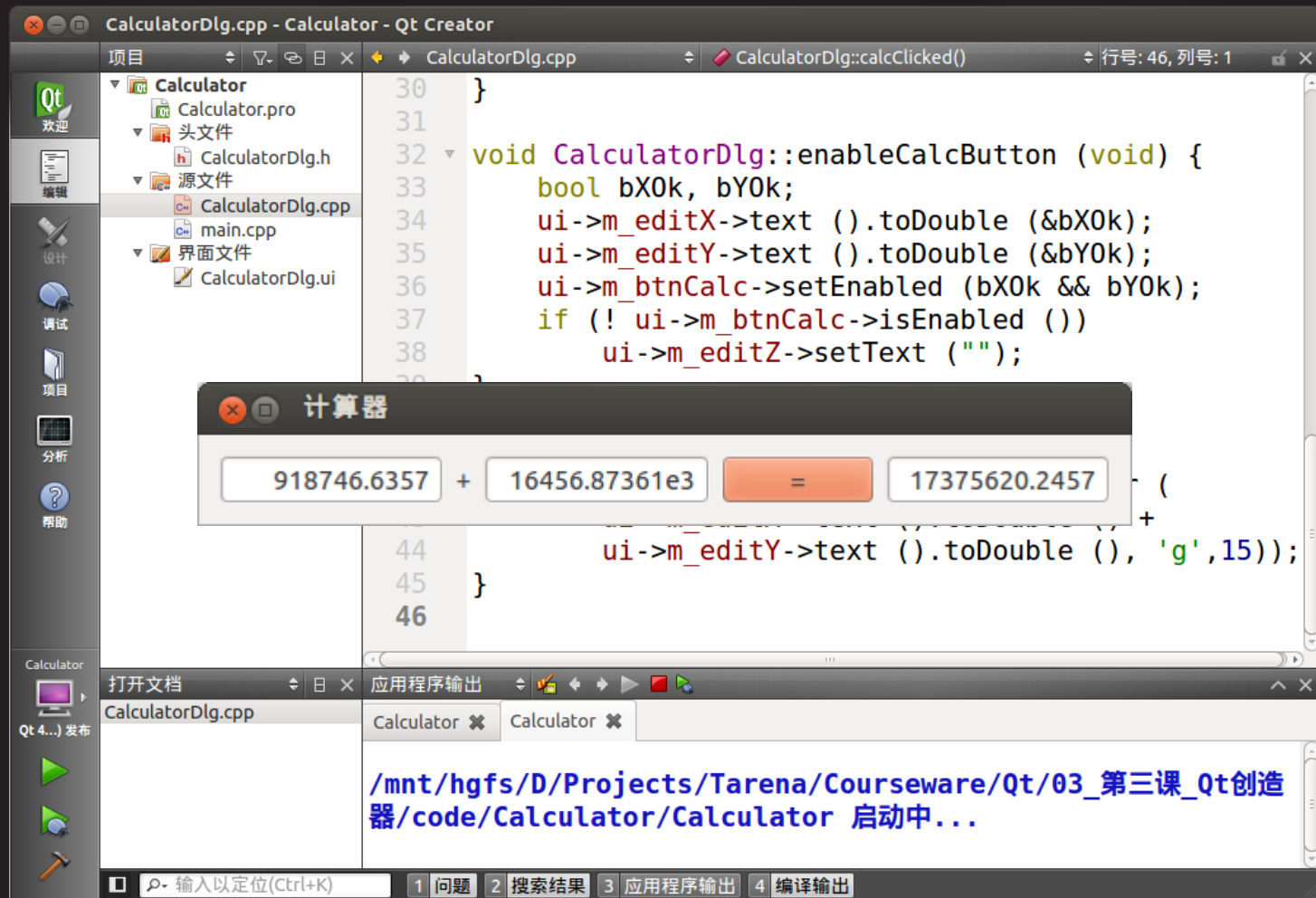
```

24         this, SLOT (calcClicked (void)));
25     }
26
27     CalculatorDlg::~CalculatorDlg()
28     {
29         delete ui;
30     }
31
32     void CalculatorDlg::enableCalcButton (void) {
33         bool bXOk, bYOk;
34         ui->m_editX->text ().toDouble (&bXOk);
35         ui->m_editY->text ().toDouble (&bYOk);
36         ui->m_btnCalc->setEnabled (bXOk && bYOk);
37         if (! ui->m_btnCalc->isEnabled ())
38             ui->m_editZ->setText ("");
39     }
40
41     void CalculatorDlg::calcClicked (void) {
42         ui->m_editZ->setText (QString::number (
43             ui->m_editX->text ().toDouble () +
44             ui->m_editY->text ().toDouble (), 'g',15));
45     }
46
  
```



在窗口子类中添加定制操作（续3）

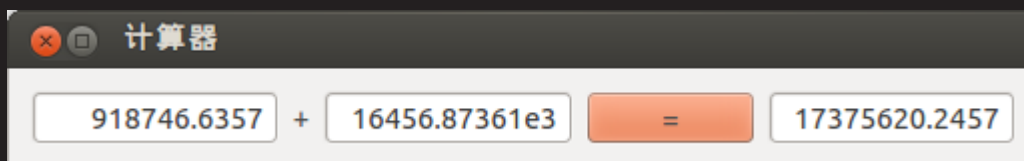
- 点击"运行"按钮或"构建/运行"菜单，启动编译、链接，如无错误，程序会被启动运行，输入数据，测试验证



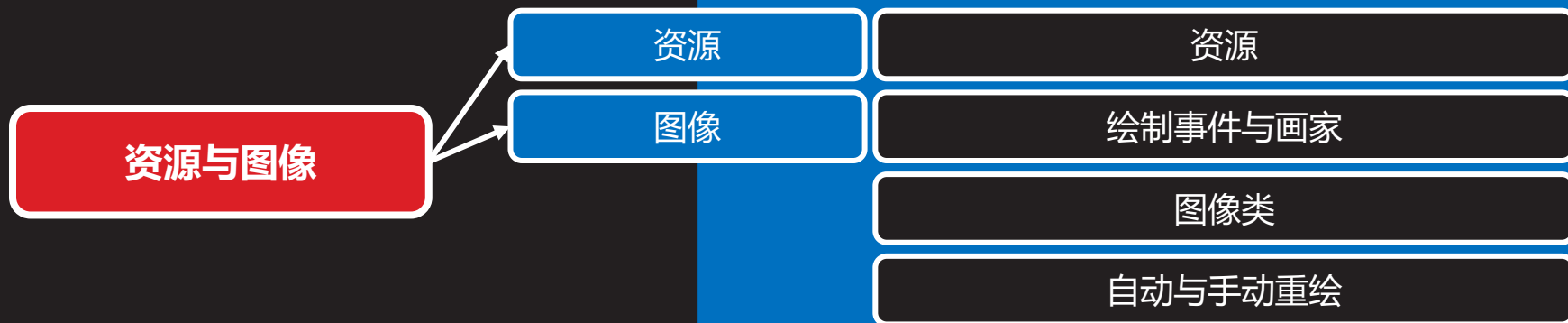
利用Qt创造器重构计算器

【参见：TTS COOKBOOK】

- 利用Qt创造器重构计算器



资源与图像



资源



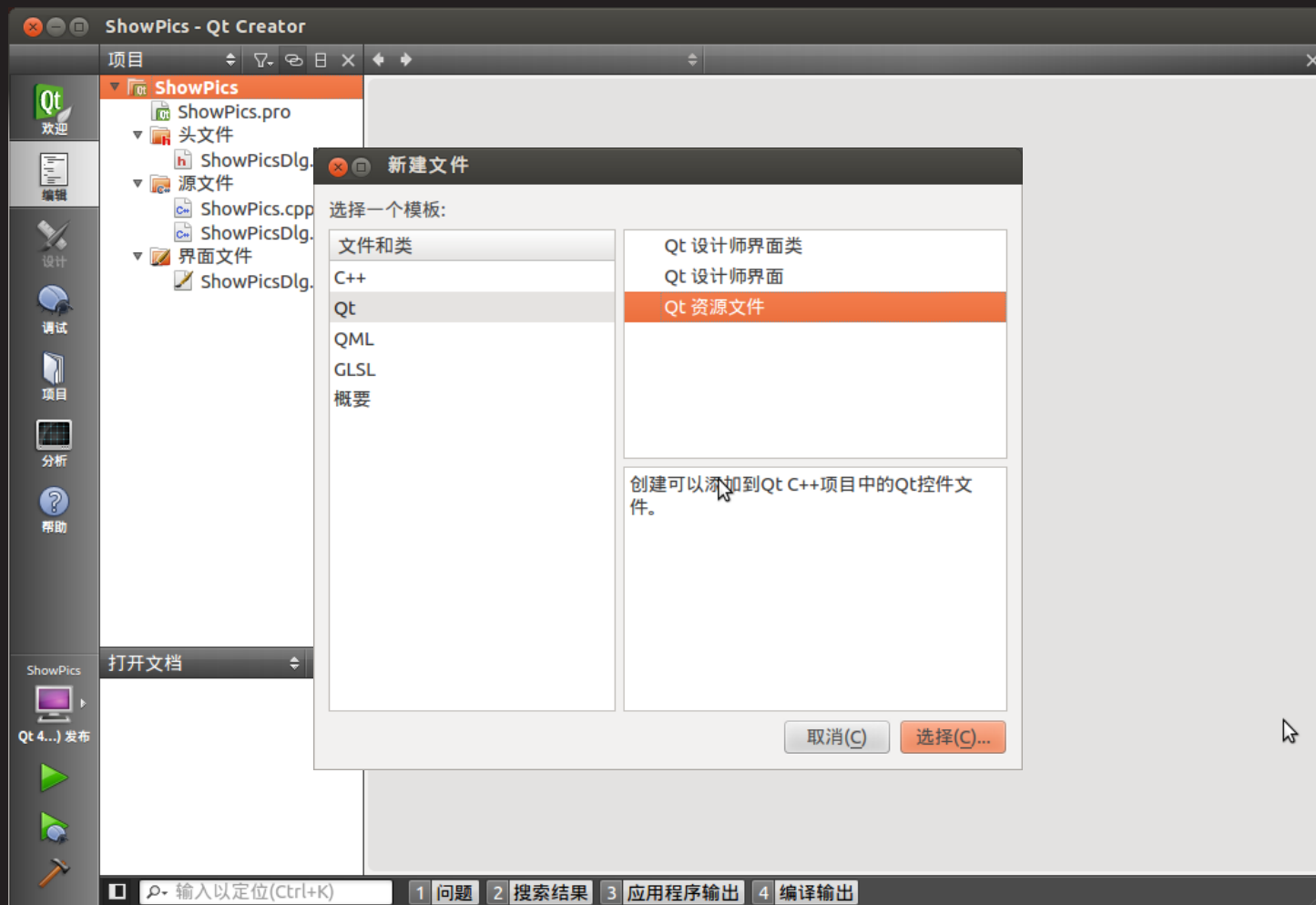
资源

- 利用Qt创造器和资源编译器可以很方便地在工程中添加**图像、音视频**等多媒体资源，并与应用程序结合在一起
- 事实上Qt的资源由两部分组成
 - **媒体文件**：图像、音视频文件本身，如：logo.jpg
 - **资源脚本**：用于描述资源的XML文件，如：ShowPics.qrc
- C++编译器无法理解二进制形式的媒体文件和XML格式的资源脚本，因此需要先用Qt提供的资源编译器rcc将资源脚本连同它所引用的媒体文件一起编译成C++源文件
 - **rcc -name ShowPics ShowPics.qrc -o qrc_ShowPics.cpp**
- 资源数据作为C++代码的一部分被**嵌入**到可执行程序中
- 以上编译过程已被包含在qmake生成的**Makefile**脚本中



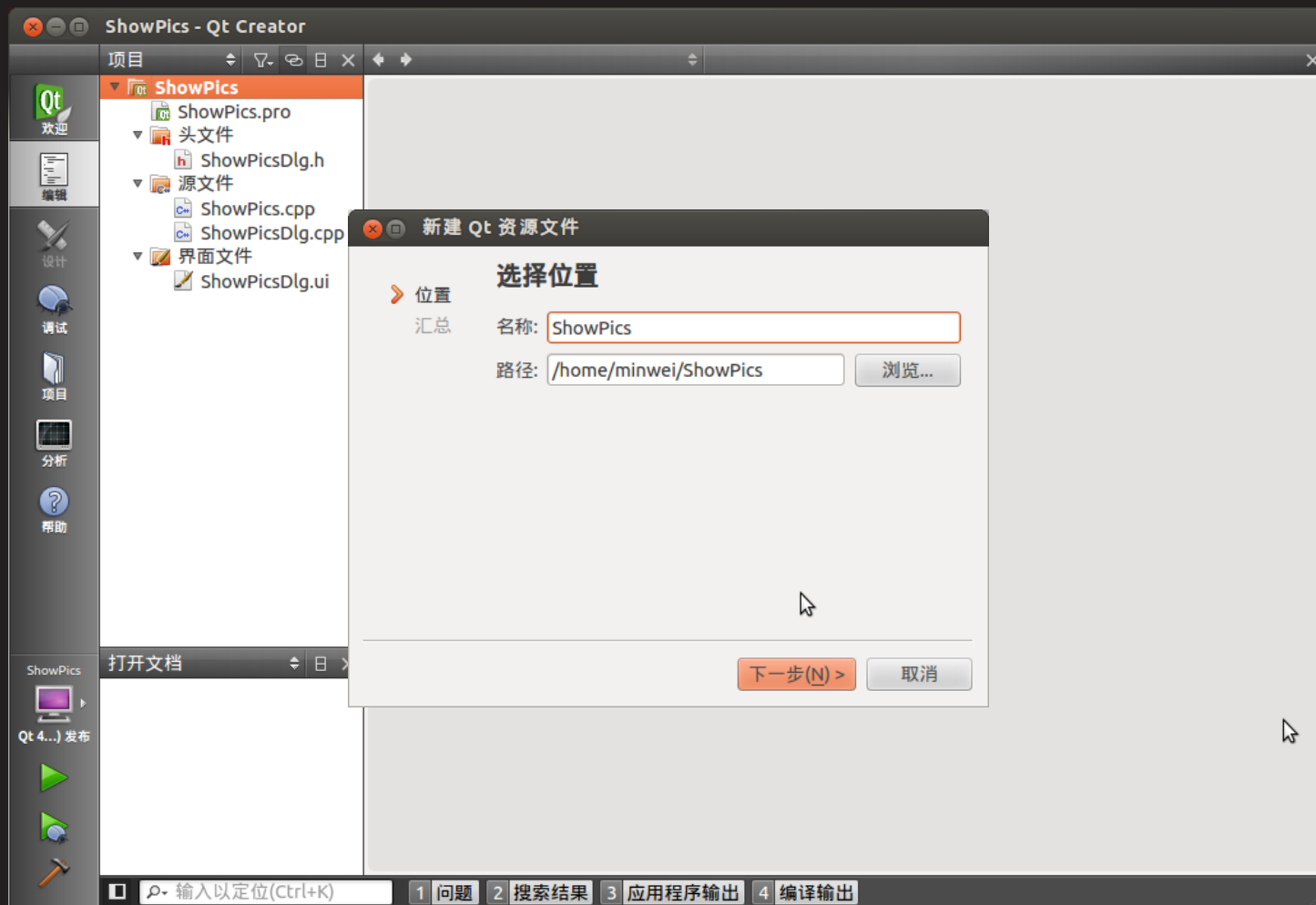
资源（续1）

- 右键单击项目名称，选择弹出菜单“添加新文件...”，然后依次选择“Qt”和“Qt资源文件”作为模板，并点击“选择...”



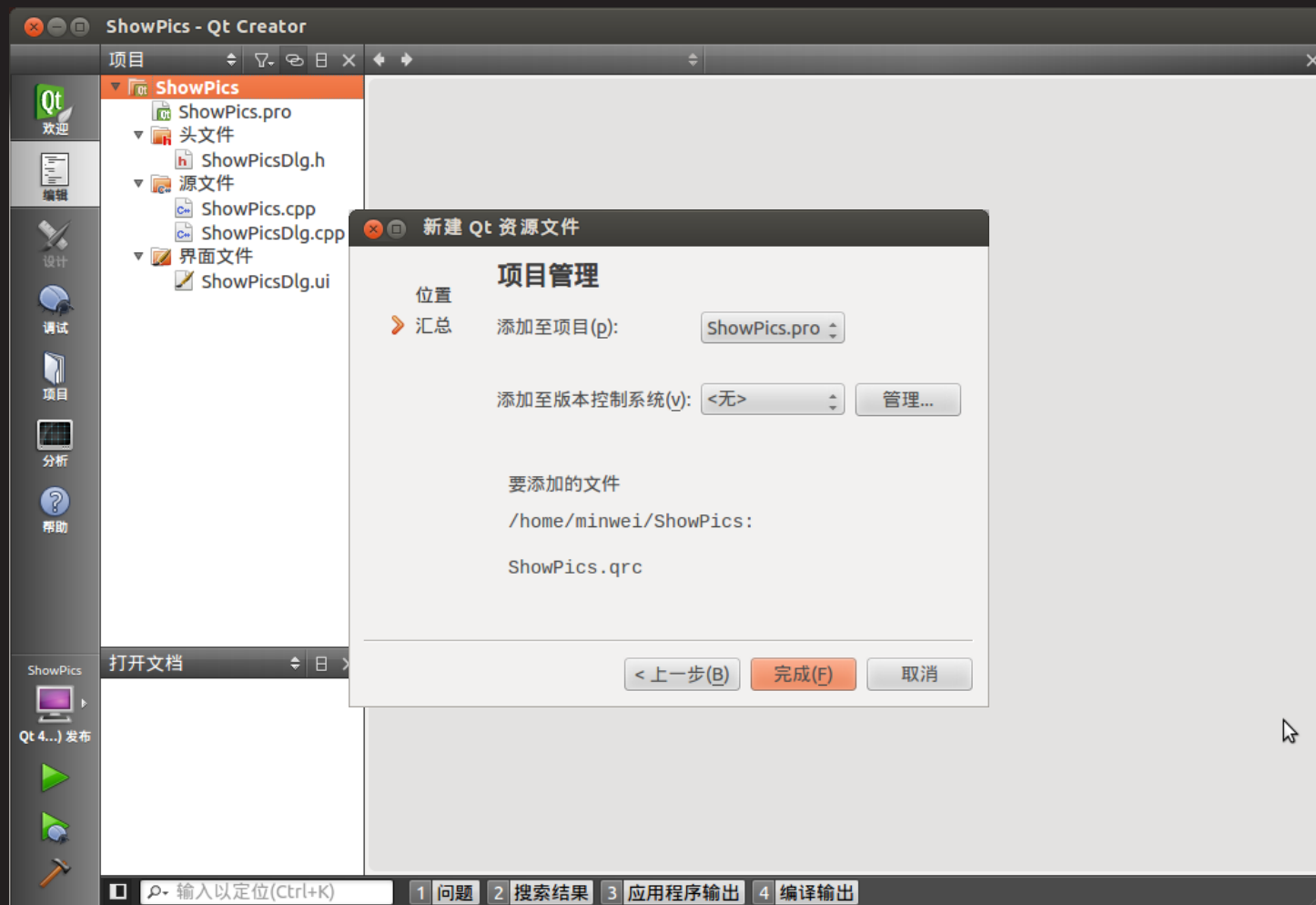
资源（续2）

- 指定资源脚本文件的名称和路径，建议和项目中的其它文件一起放在工程目录下，以便于管理，点击“下一步”



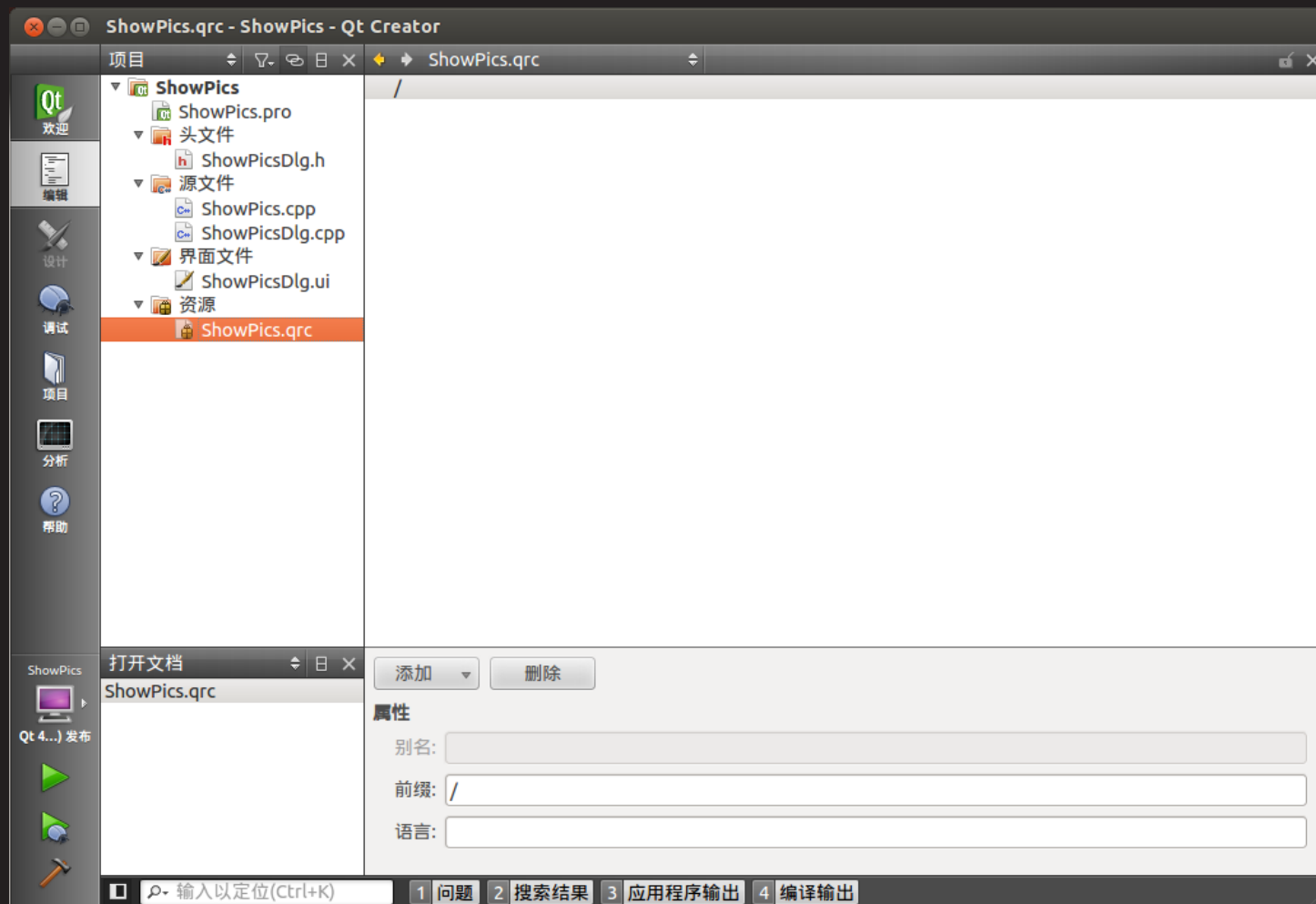
资源 (续3)

- 将所建资源脚本文件添加到**项目**(缺省选项)和**版本控制**(如果有的话)中, 点击"完成"



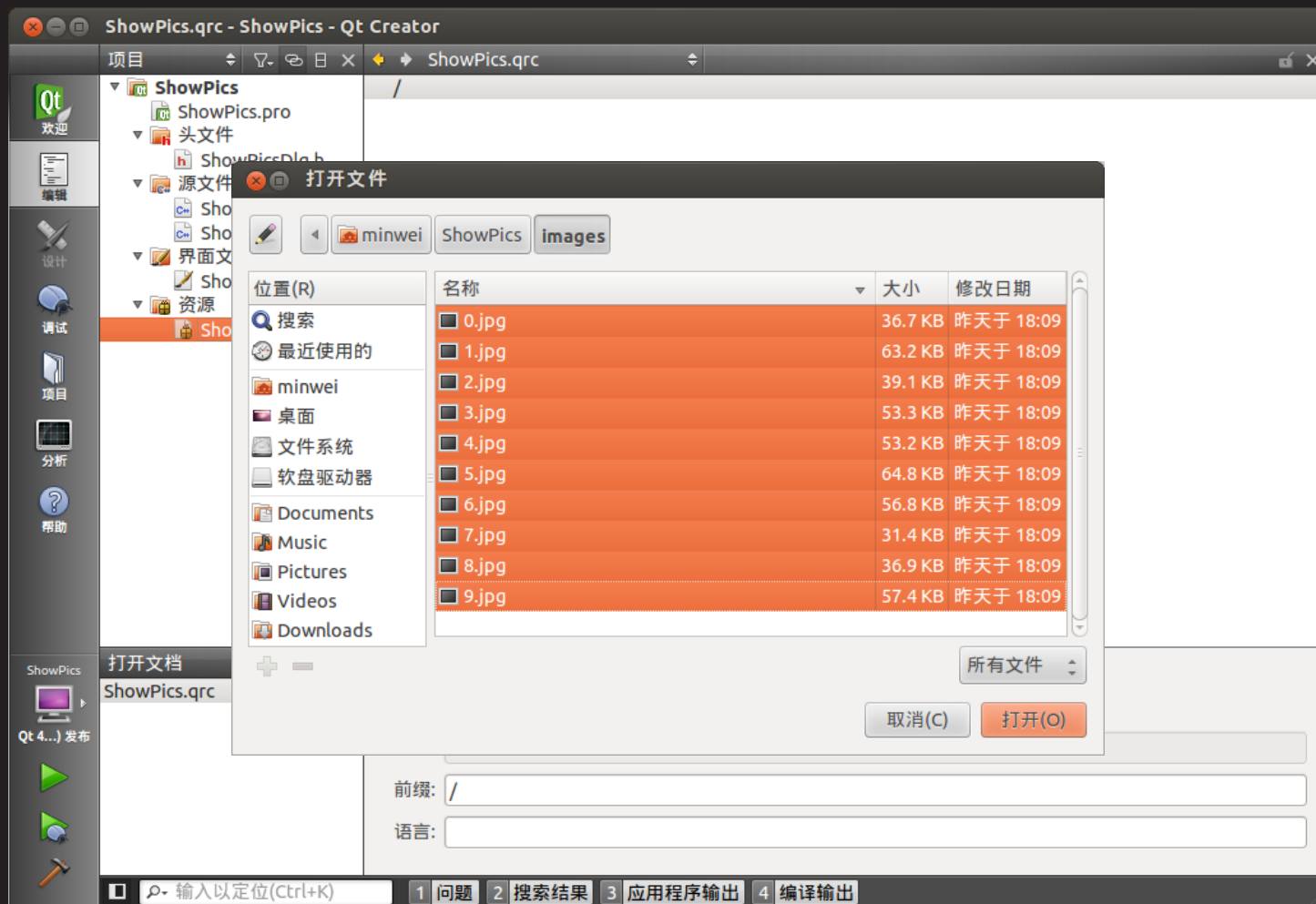
资源（续4）

- 选择菜单"添加/添加前缀"，缺省的资源前缀是"/new/prefix1"，可将其改为"/"



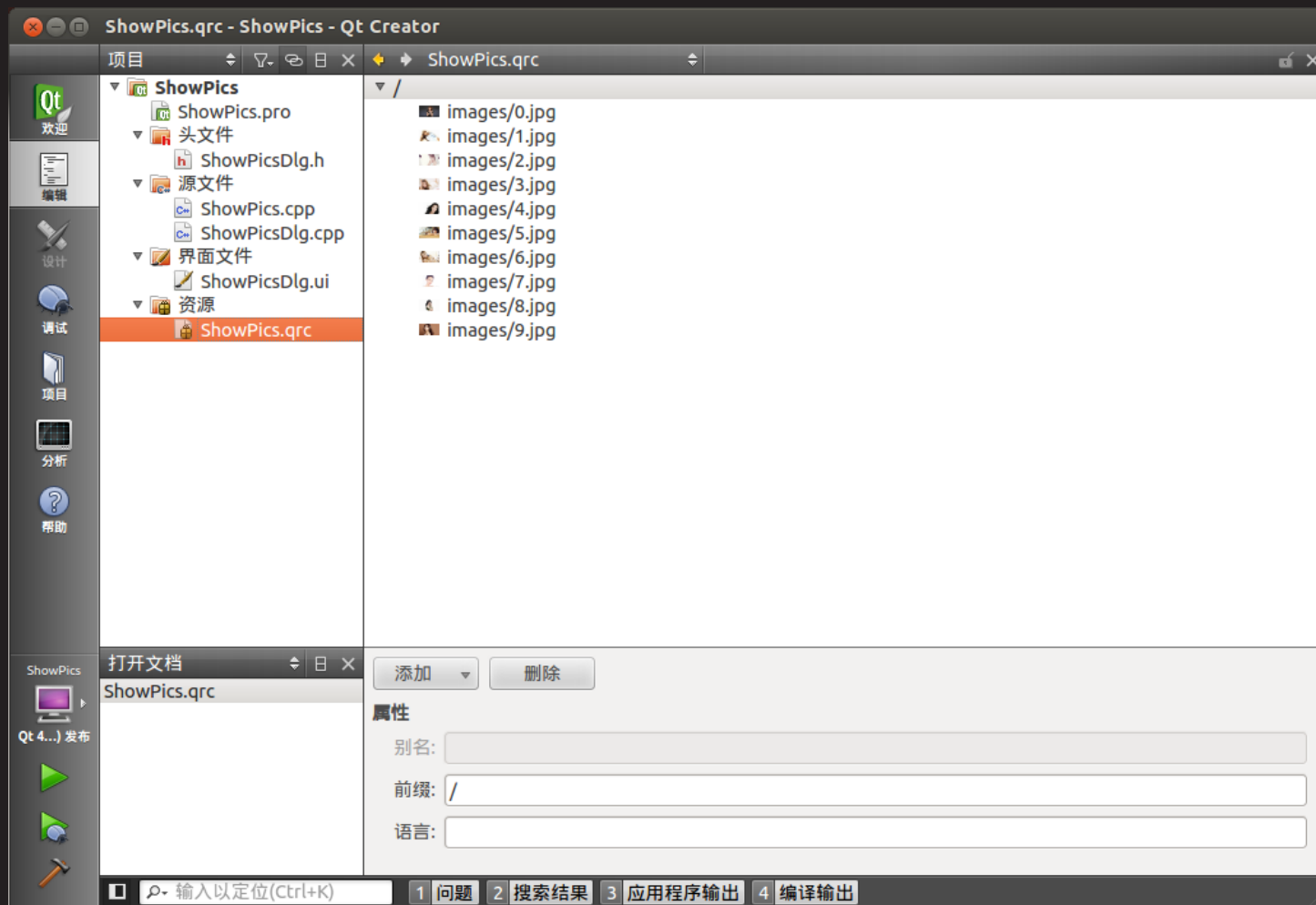
资源 (续5)

- 选择菜单"添加/添加文件", 在"打开文件"对话框中选择(可多选)需要添加的媒体文件, 点击"打开"



资源（续6）

- 程序可以`"/images/0.jpg"`的形式访问这些资源，其中第一个`"/`表示资源前缀，第二个`"/`表示目录

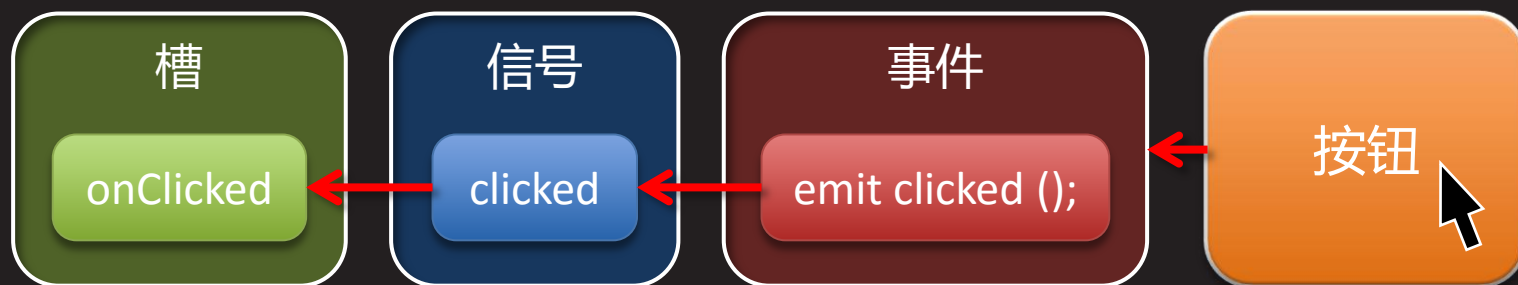


图像



绘制事件与画家

- 什么是事件？
 - 事件是由窗口系统或者Qt自身产生的，用以响应所发生的各类事情，比如用户按下并释放了键盘或者鼠标、窗口因暴露而需要重绘、定时器到期而应有所动作，等等
 - 从某种意义上讲，事件比信号更原始，甚至可以认为大多数信号其实都是由事件产生的。比如一个下压式按钮首先感受到的是鼠标事件，在进行必要的处理以产生按钮下沉继而弹起的视觉效果之后，才会发射clicked()信号



绘制事件与画家（续1）

- 如何处理事件？
 - 当事件发生时，首先被调用的是QObject类中的虚函数event()，其QEvent型参数标识了具体的事件类型
 - ✓ `bool QObject::event (QEvent* e);`
 - 作为QObject类的子类，QWidget类覆盖了其基类中的event()虚函数，并根据具体事件调用具体事件处理函数
 - ✓ `void QWidget::mousePressEvent (QMouseEvent* e);`
 - ✓ `void QWidget::mouseReleaseEvent (QMouseEvent* e);`
 - ✓ `void QWidget::paintEvent (QPaintEvent* e);`
 - 而这些事件处理函数同样也是虚函数，也可以被QWidget类的子类覆盖，以提供针对不同窗口部件类型的事件处理
 - 部件的使用者所关心的往往是定义什么样的槽处理什么样的信号，而部件的实现者则更关心覆盖哪些事件处理函数



绘制事件与画家（续2）

- 当有下列情况之一发生时，窗口部件会收到绘制事件，即QWidget类的paintEvent()虚函数会被调用
 - 窗口被创建以后第一次显示出来
 - 窗口由隐藏状态转变为可见状态
 - 窗口由最小化状态转变为正常或最大化状态
 - 窗口超出屏幕边界的区域进入屏幕范围之内
 - 窗口被遮挡的区域因某种原因重新暴露出来
 - 窗口因尺寸大小的变化需要呈现更多的内容
 - QWidget类的update()成员函数被调用
- 作为QWidget类的子类，可以在对该虚函数的覆盖版本中实现诸如显示文本、绘制图形、渲染图像等操作
 - void ShowPicsDlg::paintEvent (QPaintEvent* e) { ... }

绘制事件与画家（续3）

- QPainter类是Qt的二维图形引擎，该类具有如下功能

- 绘制矢量文字
- 绘制几何图形
- 绘制像素映射和图像
- 反走样、像素混合、渐变和矢量路径
- 平移、旋转、错切、缩放等线性变换



- QPainter类通过构造函数接收绘制设备，即在什么上画
 - QPainter::QPainter (QPaintDevice* device);
- QPainter类用于渲染图像的众多成员函数之一
 - void QPainter::drawImage (const QRect& rect, const QImage& image);

QImage

- 图形图像处理的难点往往并不在于算法的复杂性，而是如何在速度与精度之间进行取舍
 - 如果直接在QWidget上进行绘制，速度往往是最快的，但精度则完全依赖于平台自带的渲染引擎，比如Linux的X11或者Windows的GDI，相同代码的渲染效果因之大相径庭
 - 当精度重于速度时，不妨把所有内容都绘制在QImage上，最后再将渲染结果一次性复制到屏幕。这样可以总是使用Qt内置的渲染引擎，以在所有平台上获得相同的显示效果
- QImage已经内置了针对不同格式图像的处理算法
 - BMP、GIF、JPG、JPEG、PNG、PBM、PGM、PPM、TIFF、XBM、XPM
- QImage既可以从文件也可以从嵌入式资源中加载图像
 - QImage::QImage (const char* fileName, const char* format = 0);

自动与手动重绘

- 借助绘制事件、QPainter和QImage在窗口中显示图像
 - void ShowPicsDlg::paintEvent (QPaintEvent* e) {


```

          QPainter painter (this);
          QRect rclImage = ui->m_frmlImage->frameRect ();
          rclImage.translate (ui->m_frmlImage->pos ());
          QImage image (":/images/" +
            QString::number (m_idxImage) + ".jpg");
          painter.drawImage (rclImage, image);
          }
          
```
- 窗口绘制除了由系统自动触发外，也可以人为手动触发
 - void QWidget::update (void); // 引发绘制事件
 - void QWidget::repaint (void); // 直接调用paintEvent()



基于资源的图片浏览器

【参见：TTS COOKBOOK】

- 基于资源的图片浏览器



目录与定时器



目录

QVector

- **容器**是用于在内存中存储给定类型的若干元素的类模板
- C++已经提供了许多容器，并作为标准模板库(STL)的一部分，被包含在标准C++库中，谓之**标准容器**
- Qt提供了自己的容器类，相比于标准容器，**Qt容器**在所有平台上运行时都具有完全一致的表现
- **QVector<T>**是一种类似数组的线性表容器，连续内存的物理结构决定了它具有常数时间的随机访问能力
 - **动态**内存管理和静态初始化
 - 支持数组式的随机**下标**访问
 - 支持插入运算以在尾端**追加**
 - 支持线性时间的**插入**和**删除**



QDir

- QDir类提供了一种平台无关的遍历目录的方法。例如，下面的代码将在一个给定的路径下加载所有格式的照片

```
– void ErnieDlg::loadPhotos (QString const& path) {  
    QStringList filters;  
    foreach (QByteArray format,  
        QImageReader::supportedImageFormats ())  
        filters += "*" + format;  
    QDir dir (path);  
    foreach (QString file, dir.entryList (filters, QDir::Files)) {  
        QString name = file;  
        name.resize (name.lastIndexOf ('.') + 1);  
        m_vecPhotos << qMakePair (name,  
            QImage (path + QDir::separator () + file));  
    }  
    foreach (QString subDir, dir.entryList (  
        QDir::Dirs | QDir::NoDotAndDotDot))  
        loadPhotos (path + QDir::separator () + subDir);  
}
```



定时器



定时器

- Qt通过两套机制为应用程序提供定时功能
 - 定时器**事件**，由QObject提供
 - 定时器**信号**，由QTimer提供
- 通过定时器事件实现定时
 - `int QObject::startTimer (int interval);`
 - ✓ 启动定时器，以后每隔interval毫秒触发一次定时器事件
 - ✓ 若interval取0，则只要没有其它事件，就触发定时器事件
 - ✓ 成功返回定时器ID，失败返回0
 - `void QObject::timerEvent (QTimerEvent* e) [virtual];`
 - ✓ 定时器事件处理函数，QTimerEvent的timerId()成员函数返回触发该次事件的定时器ID
 - `void QObject::killTimer(int id);`
 - ✓ 销毁参数id所标识的定时器，此后再无源自该定时器的定时事件



基于目录和定时器的摇奖机

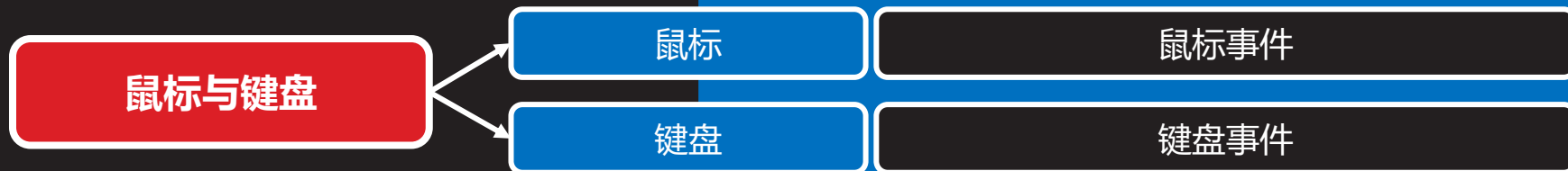
【参见：TTS COOKBOOK】

课堂练习

- 基于目录和定时器的摇奖机



鼠标与键盘



鼠标



鼠标事件

- QWidget类定义了以下虚函数提供对鼠标事件的处理，分别响应鼠标键的按下、弹起和双击，以及鼠标移动
 - virtual void mousePressEvent (QMouseEvent* e);
 - virtual void mouseReleaseEvent (QMouseEvent* e);
 - virtual void mouseDoubleClickEvent (QMouseEvent* e);
 - virtual void mouseMoveEvent (QMouseEvent* e);
- QMouseEvent类提供了有关鼠标事件的具体细节
 - Qt::MouseButton button (void) const; // 引发事件的鼠标键
 - Qt::MouseButtons buttons (void) const; // 其它鼠标键状态
 - const QPoint& pos (void) const; // 窗口坐标鼠标位置
 - const QPoint& globalPos (void) const; // 屏幕坐标鼠标位置

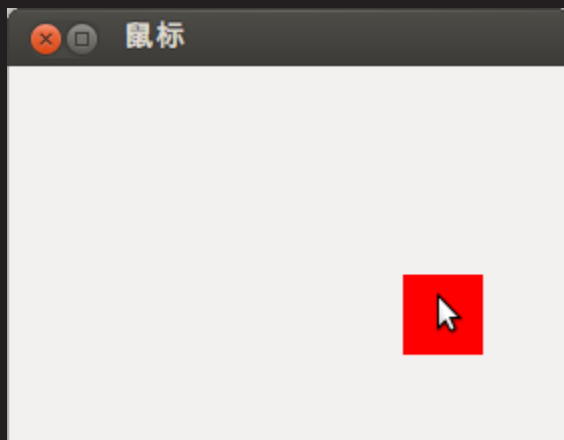


鼠标事件处理

【参见：TTS COOKBOOK】

课堂练习

- 鼠标事件处理



键盘



键盘事件

- QWidget类定义了以下虚函数提供对键盘事件的处理，分别响应键盘按键的按下和弹起
 - virtual void keyPressEvent (QKeyEvent* e);
 - virtual void keyReleaseEvent (QKeyEvent* e);
- QKeyEvent类提供了有关键盘事件的具体细节
 - int count (void) const; // 按键次数
 - bool isAutoRepeat (void) const; // 自动重复
 - int key (void) const; // 按键代码
 - Qt::KeyboardModifiers modifiers (void) const; // 组合按键
 - quint32 nativeScanCode (void) const; // 扫描键码
 - quint32 nativeVirtualKey (void) const; // 虚拟键码
 - QString text (void) const; // 按键文本



键盘事件处理

【参见：TTS COOKBOOK】

- 键盘事件处理



The screenshot shows a terminal window titled "minwei@ubuntu: Keyboard". The output text is as follows:

```
键盘弹起
按键次数：1
自动重复：否
按键代码：65
组合按键：
扫描键码：38
虚拟键码：97
按键文本：a
```



贪食蛇游戏



准备工作



QTimer和QList

- QTimer类提供了一套高级定时器编程接口。基于这套接口可以直接创建QTimer对象，并将它的timeout()信号与适当的槽连接，通过该类的start()成员函数，在启动定时器的同时，提供以毫秒为单位的定时间隔，每经过一个这样的间隔，定时器对象就会发射timeout()信号
 - QTimer* timer = new QTimer (this);
connect (timer, SIGNAL (timeout (void)),
this, SLOT (onTimeout (void)));
timer->start (1000);
- QList容器的物理结构是一个数组链表，它结合了完全连续内存的QVector和完全离散内存的QLinkedList的双重优点，既支持基于下标的随机访问，又能快速地在任意位置插入删除。QList被认为是最具一般性的多用途容器

属性、方法和事件



属性

- 属性描述对象的**状态**，一方面表示对象的逻辑模型，另一方面又通过其自身的变化表示对象的活动与交互过程

类型	名称	描述
QList<QLabel*>	m_lstSnake	蛇节点链表
QLabel*	m_labFood	食物
EDIR	m_eDir	行进方向
static const int	s_nStep = 20	步距
unsigned int	m_uScore	得分
QImage	m_imgAction	活动区图片
QTimer	m_tmrCrawl	爬行定时器

- 属性多具有名词或形容词性，通常被实现为类的**成员变量**。其中一些可能带有常属性，而另一些则带有静态性



方法

- 方法描述对象的**行为**，一些方法与属性无关，而另一些方法或以属性为前提，或以属性为后果，抑或兼而有之

访问	名称	描述
外部	SnakeDlg	构造函数
外部	~SnakeDlg	析构函数
内部	initGame	初始化游戏
内部	makeFood	制造食物
内部	nextStep	计算下一步
内部	validPos	判断有效位置
内部	updateScoreboard	更新记分板
内部	rand	生成给定值域内的随机数
内部	round	对齐圆整

- 方法多具有动词性，通常被实现为类的**成员函数**(含静态)



事件

- 事件类似于方法，但较后者更强调其**被动性**。方法是按过程依次触发的行为，而事件则是面向诸可能性的应对

访问	名称	描述
外部	showEvent	显示
外部	paintEvent	绘制
外部	keyPressEvent	按键

- 在类似Qt这样应用框架中，事件多被实现为基类的**虚函数**甚至纯虚函数，由子类在其**覆盖**版本中提供实际功能
- 游戏在**显示**事件处理中执行诸如启动定时器之类的初始化，在**绘制**事件处理中为活动区贴上图片，根据每次所按**方向键**的不同，更新**m_eDir**的值，更新蛇的行进方向



信号和槽



信号和槽

- 信号从某种意义上讲也是一种**事件**，同样代表了那些不可预期的可能性，而槽正是针对这种可能性的**被动**方法

信号	槽
<code>void QTimer::timeout (void) [signal]</code>	<code>void SnakeDlg::crawlExpired (void)</code>

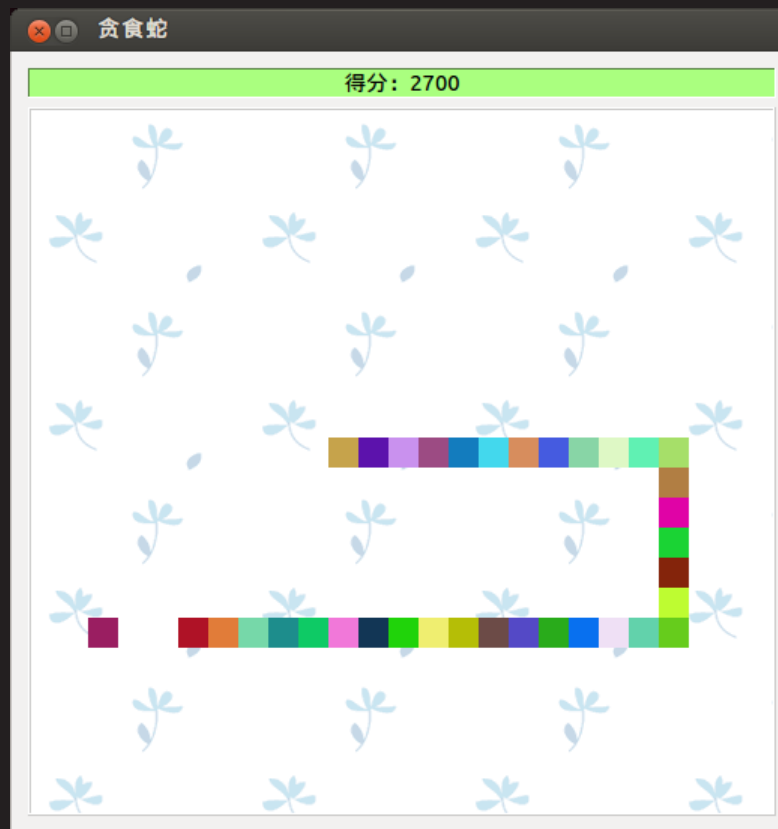
- 槽函数通常被声明为类的公有成员函数，但如果与信号的连接在声明槽的类的内部完成，则**私有槽**也是允许的
- 每次定时器到期都要决定蛇的下一个状态
 - 若下一步是**有效**位置且是**食物**，则生长并加分
 - 若下一步是**有效**位置但**非食物**，则爬到该位置
 - 若下一步是**无效**位置，则提示用户重来或退出



贪食蛇游戏

【参见：TTS COOKBOOK】

- 贪食蛇游戏



总结和答疑

