



Qt程序设计方法

QT基础

**School of Computer Science
and Engineering**

2024-2025-暑期课程



国际化、帮助系统和Qt插件

主要内容

- 国际化
- 帮助系统
- 创建Qt插件



国际化

国际化的英文表述为Internationalization，通常简称为I18N（首尾字母加中间的字符数），一个应用程序的国际化就是使该应用程序可以让其他国家的用户使用的过程。Qt支持现在使用的大多数语言，特别是：

- 所有东亚语言（汉语、日语和朝鲜语）
- 所有西方语言（使用拉丁字母）
- 阿拉伯语
- 西里尔语言（俄语和乌克兰语等）
- 希腊语
- 希伯来语
- 泰语和老挝语
- 所有在Unicode 6.2中不需要特殊处理的脚本

在Qt中，所有的输入部件和文本绘制方式对Qt所支持的所有语言都提供了内置的支持。Qt内置的字体引擎可以在同一时间正确而且精细的绘制不同的文本，这些文本可以包含来自众多不同书写系统的字符。



使用Qt Linguist翻译应用程序

在Qt中编写代码时要对需要显示的字符串调用tr()函数，完成代码编写后，对这个应用程序的翻译主要包含三步：

- 1.运行lupdate工具从C++源代码中提取要翻译的文本，这时会生成一个.ts文件，这个文件是XML格式的；
- 2.在Qt Linguist中打开.ts文件，并完成翻译工作；
- 3.运行lrelease工具从.ts文件中获得.qm文件，它是一个二进制文件。这里的.ts文件是供翻译人员使用的，而在程序运行时只需要使用.qm文件，这两个文件都是与平台无关的。



第一步：编写源码

```
QLabel *label = new QLabel(this);
```

```
label->setText
```

```
label->move(1
```

```
QLabel *label2
```

```
label2->setTex
```

```
label2->move(
```

```
QLabel *label3
```

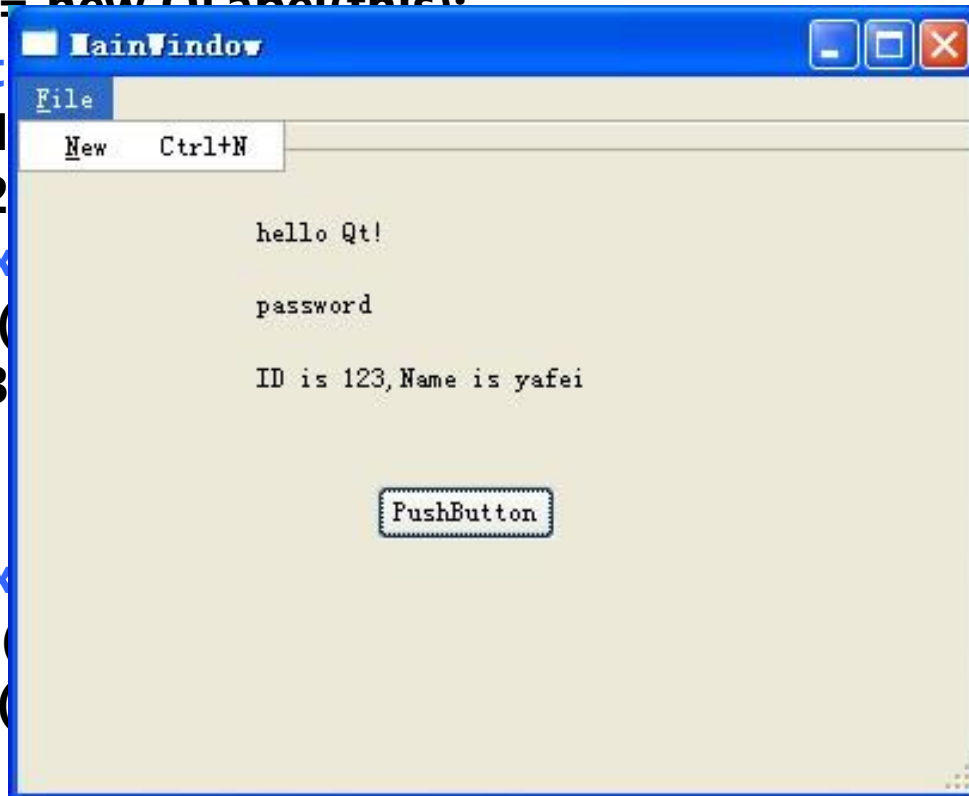
```
int id = 123;
```

```
QString name
```

```
label3->setTex
```

```
label3->resize(
```

```
label3->move(
```



```
name));
```

这里向界面上添加了三个标签，因为这三个标签中的内容都是用户可见的，所以需要调用tr()函数。



tr()函数

tr()函数一共有三个参数，它的原型如下：

```
QString QObject::tr ( const char * sourceText,  
                     const char * disambiguation = 0,  
                     int n = -1 ) [static]
```

- 第一个参数sourceText就是要显示的字符串，tr()函数会返回sourceText的译文；
- 第二个参数disambiguation是消除歧义字符串，比如这里的password，如果一个程序中需要输入多个不同的密码，那么在没有上下文的情况下，就很难确定这个password到底指哪个密码。这个参数一般使用类名或者部件名，比如这里使用了mainwindow，就说明这个password是在mainwindow上的；
- 第三个参数n表明是否使用了复数，因为英文单词中复数一般要在单词末尾加“s”，比如“1 message”，复数时为“2 messages”。遇到这种情况，就可以使用这个参数，它可以根据数值来判断是否需要添加“s”，例如：
int n = messages.count();
showMessage(tr("%n message(s) saved", "", n));



第二步，更改项目文件

要在项目文件中指定生成的.ts文件，每一种翻译语言对应一个.ts文件。
打开myl18N.pro文件，在最后面添加如下一行代码：

```
TRANSLATIONS = myl18N_zh_CN.ts
```

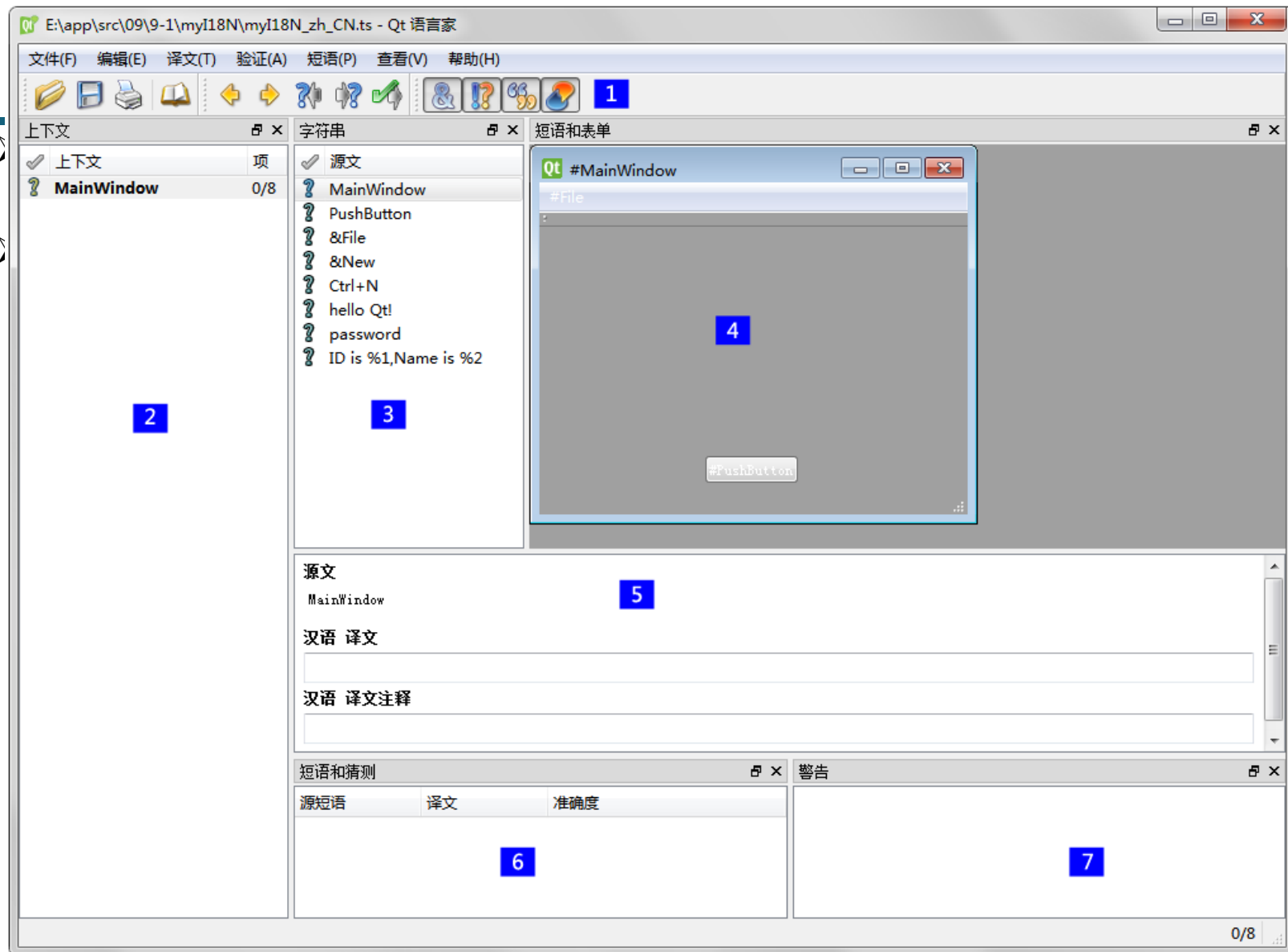
这表明后面生成的.ts文件的文件名为“myl18N_zh_CN.ts”，对于.ts的名称可以随意编写，不过一般是以区域代码来结尾，这样可以更好的区分，例如这里使用了“zh_CN”来表示简体中文。**最后按下Ctrl+S保存该文件。**



第三步，使用lupdate生成.ts文件

- 当要进行翻译工作时，先要使用lupdate工具来提取源代码中的翻译文本，生成.ts文件。
- 单击“工具→外部→Qt语言家→更新翻译(lupdate)”菜单项，从概要信息输出栏中可以看到，更新了“myl18N_zh_CN.ts”文件，发现了8个源文本，其中有8条新的翻译和0条已经存在的翻译。这表明可以对程序代码进行更改，然后多次运行lupdate，而只需要翻译新添加的内容。
- 可以在项目目录中使用写字板打开这个.ts文件，可以看到它是XML格式的，其中记录了字符串的位置和是否已经被翻译等信息。














字符串
在弹



①菜单项和工具栏

在菜单栏中列出了Qt Linguist的所有功能选项，而工具栏中列出了常用的一些功能，其中后面9个图标的功能为：

-  在字符串列表中移动到前一个条目
-  在字符串列表中移动到下一个条目
-  在字符串列表中移动到前一个没有完成翻译的条目
-  在字符串列表中移动到下一个没有完成翻译的条目
-  标记当前条目为完成翻译状态，然后移动到下一个没有完成翻译的条目
-  打开或关闭加速键（accelerator）验证（validation）。打开加速键验证可以验证加速键是否被翻译，例如字符串中包含“&”符号，但是翻译中没有包含“&”符号，则验证失败。
-  打开或关闭短语结束标点符号验证。打开短语标点符号验证可以验证翻译中是否使用了和字符串中相同的标点来结尾。
-  打开或关闭短语参考（phrase book）验证。打开短语参考验证可以验证翻译是否和短语参考中的翻译相同。在翻译相似的程序时，若希望将常用的翻译记录下来，以便以后使用，就可以使用短语参考。可以通过“Phrases” → “New Phrase Book”菜单来创建一个新的短语参考，然后再翻译字符串时使用Ctrl+T将这个字符串及其翻译放入短语参考中。
-  打开或关闭占位符（place marker）验证。打开占位符验证可以验证翻译中是否使用了和字符串中相同的占位符，例如%1，%2等。



②上下文（Context）窗口

这里是一个上下文列表，罗列了要翻译的字符串所在位置的上下文。其中的“Context”列使用字母表顺序罗列了上下文的名字，它一般是QObject子类的名字；而“Item”列显示的是字符串数目，例如0/8表明有8个要翻译的字符串，已经翻译了0个。在每个上下文的最左端用图标表明了翻译的状态，它们的含义是：

- ✔ 上下文中的所有字符串都已经被翻译，而且所有的翻译都通过了验证测试（validation test）；
- ✔ 上下文中的所有字符串或者都已经被翻译，或者都已经标记为已翻译，但是至少有一个翻译验证测试失败；
- ？ 在上下文中至少有一个字符串没有被翻译或者没有被标记为已翻译；
- ✔ 在该上下文中没有再出现要翻译的字符串，这通常意味着这个上下文已经不在应用程序中了。



③字符串 (String) 窗口

这里罗列了在当前上下文中找到的所有要翻译的字符串。在这里选择一个字符串，可以使这个字符串在翻译区域进行翻译。在字符串左边使用图标表明了字符串的状态，它们的含义是：

- ✔ 源字符串已经翻译（可能为空），或者用户已经接受翻译，而且翻译通过了所有验证测试；
- ✔ 用户已经接受了翻译，但是翻译没有通过所有的验证测试；
- ？ 字符串已经拥有了一个通过了所有验证测试的非空翻译，但是用户还没有接受该翻译；
- ？ 字符串还没有翻译；
- ！ 字符串拥有一个翻译，但是这个翻译没有通过所有的验证测试；
- ✔ 字符串已经过时，它已经不在该上下文中。



- ④**短语和表单 (Sources and Forms) 窗口**：如果包含有要翻译字符串的源文件在Qt Linguist中可用，那么这个窗口会显示当前字符串在源文件中的上下文。
- ⑤**翻译区域 (The Translation Area)**：在字符串列表中选择的字符串会出现在翻译区域的最顶端的“Source Text”下面；如果在使用tr()函数时设置了第二个参数消除歧义注释，那么这里还会在“Developer comments”下出现该注释；后面的“translation”中可以输入翻译文本，如果文本中包含空格，会使用“.”显示；最后面的“translator comments”中可以填写翻译注释文本。
- ⑥**短语和猜测 (Phrases and Guesses)**：如果字符串列表中的当前字符串出现在了已经加载的短语参考中，那么当前字符串和它在短语参考中的翻译会被罗列在这个窗口。在这里可以双击翻译文本，这样翻译文本就会复制到翻译区域。
- ⑦**警告信息 (Warnings) 窗口**：如果输入的当前字符串的翻译没有通过开启的验证测试，那么在这里会显示失败信息。



翻译程序

翻译区域可以看到现在已经是要翻译成汉语，这是因为我们的.ts文件名中包含了中文的区域代码。如果这里没有正确显示要翻译成的语言，那么可以使用“编辑→翻译文件设置”菜单来更改。

原文本	翻译文本
MainWindow	应用程序主窗口
PushButton	按钮
&File	文件(&F)
&New	新建(&N)
Ctrl+N	Ctrl+N
hello Qt!	你好 Qt!
password	密码
ID is %1,Name is %2	账号是%1，名字是%2



第五步，使用lrelease生成.qm文件

- 可以在Qt Linguist中使用“文件→发布”和“文件→另外发布为”这两个菜单项来生成当前已打开的.ts文件对应的.qm文件，文件会生成在.ts所在目录下。
- 还可以通过Qt Creator的“工具→外部→Qt语言家→发布翻译(lrelease)”菜单项来完成。



第六步，使用.qm文件

在项目中
文件，然后

QTransla
translat
a.install

这里先加
对象安装
比如这里
部件进行翻译。



main.cpp
如下代码：

plication
的代码之前，
样才能对该



程序翻译中的相关问题

- 对所有用户可见的文本使用QString
- 对所有文字文本使用tr()函数
- 对加速键的值使用QKeySequence()函数，例如：
`exitAct = new QAction(tr("E&xit"), this);`
`exitAct->setShortcuts(QKeySequence::Quit);`
- 对动态文本使用QString::arg()函数



帮助系统

一个完善的应用程序应该提供尽可能丰富的帮助信息。在Qt中可以使用工具提示、状态提示以及“**What's This**”等简单的帮助提示，也可以使用Qt Assistant来提供强大的在线帮助。

- 简单的帮助提示
- 定制Qt Assistant



简单的帮助提示

“What’s this?”

题栏中有一
这时如果哪
移动到它上

- 设计模式实现
‘这是什么’
在这里输入

- 代码方式实现
QAction *
ui->main



会看到在标
his” 模式，
么当鼠标
助提示。

选择“改变
舌框，可以
助提示。



定制Qt Assistant

为了将Qt Assistant定制为自己的应用程序的帮助浏览器，需要先进行一些准备工作，主要是生成一些文件，最后再在程序中启动Qt Assistant。主要的步骤如下：

- 1.创建HTML格式的帮助文档；
- 2.创建Qt帮助项目（Qt help project）.qhp文件，该文件是XML格式的，用来组织文档，并且使它们可以在Qt Assistant中使用；
- 3.生成Qt压缩帮助（Qt compressed help）.qch文件，该文件由.qhp文件生成，是二进制文件；
- 4.创建Qt帮助集合项目（Qt help collection project）.qhcp文件，该文件是XML格式的，用来生成下面的.qhc文件；
- 5.生成Qt帮助集合（Qt help collection）.qhc文件，该文件是二进制文件，可以使Qt Assistant只显示一个应用程序的帮助文档，也可以定制Qt Assistant的外观和一些功能；
- 6.在程序中启动Qt Assistant。



第一步，创建HTML格式的帮助用户文档

可以通过各种编辑器例如Microsoft Word来编辑要使用的文档，最后保存为HTML格式的文件。

例如这里我们创建了5个HTML文件。然后在项目目录中新建文件夹，命名为“documentation”，再将这些HTML文件放入其中。

再在documentation文件夹中再新建一个“images”文件夹，往里面复制一个图标图片，以后将作为Qt Assistant的图标，例如这里使用了yafeilinux.png图片。



第二步，创建.qhp文件

创建一个文本文件另存为
“myHelp.qhp”，.qhp文件是XML格式的。



```
<?xml version="1.0" encoding="GB2312"?>
```

```
<QtHelpProject version="1.0">
```

```
<namespace>yafeilinux.myHelp</namespace>
```

```
<virtualFolder>doc</virtualFolder>
```

```
<filterSection>
```

```
<toc>
```

```
<section title="我的帮助" ref="./index.html">
```

```
<section title="关于我们" ref="./aboutUs.html">
```

```
<section title="关于yafeilinux" ref="./about_yafeilinux.html"></section>
```

```
<section title="关于Qt Creator系列教程" ref="./about_QtCreator.html"></section>
```

```
</section>
```

```
<section title="加入我们" ref="./joinUs.html"></section>
```

```
</section>
```

```
</toc>
```

```
<keywords>
```

```
<keyword name="关于" ref="./aboutUs.html"/>
```

```
<keyword name="yafeilinux" ref="./about_yafeilinux.html"/>
```

```
<keyword name="Qt Creator" ref="./about_QtCreator.html"/>
```

```
</keywords>
```

```
<files>
```

```
<file>about_QtCreator.html</file>
```

```
<file>aboutUs.html</file>
```

```
<file>about_yafeilinux.html</file>
```

```
<file>index.html</file>
```

```
<file>joinUs.html</file>
```

```
<file>images/*.png</file>
```

```
</files>
```

```
</filterSection>
```

```
</QtHelpProject>
```

- 第三行指定了命名空间namespace，这里每一个命名空间都对应一个HTML文件。
- 第四行指定了虚拟文件夹virtualFolder，这个变量可以创建URL的基址。

那么编码一般是UTF-8；

- 在目录表toc（table of contents）标签中创建了所有HTML文件的目录，指定了它们的标题和对应的路径。

- 再下面的过滤器部分filterSection标签，它指定了所有索引的关键字和对应的文件。这些关键字会显示在Qt Assistant的索引页面；在files标签中列出了所有的文件，也包含图片文件。

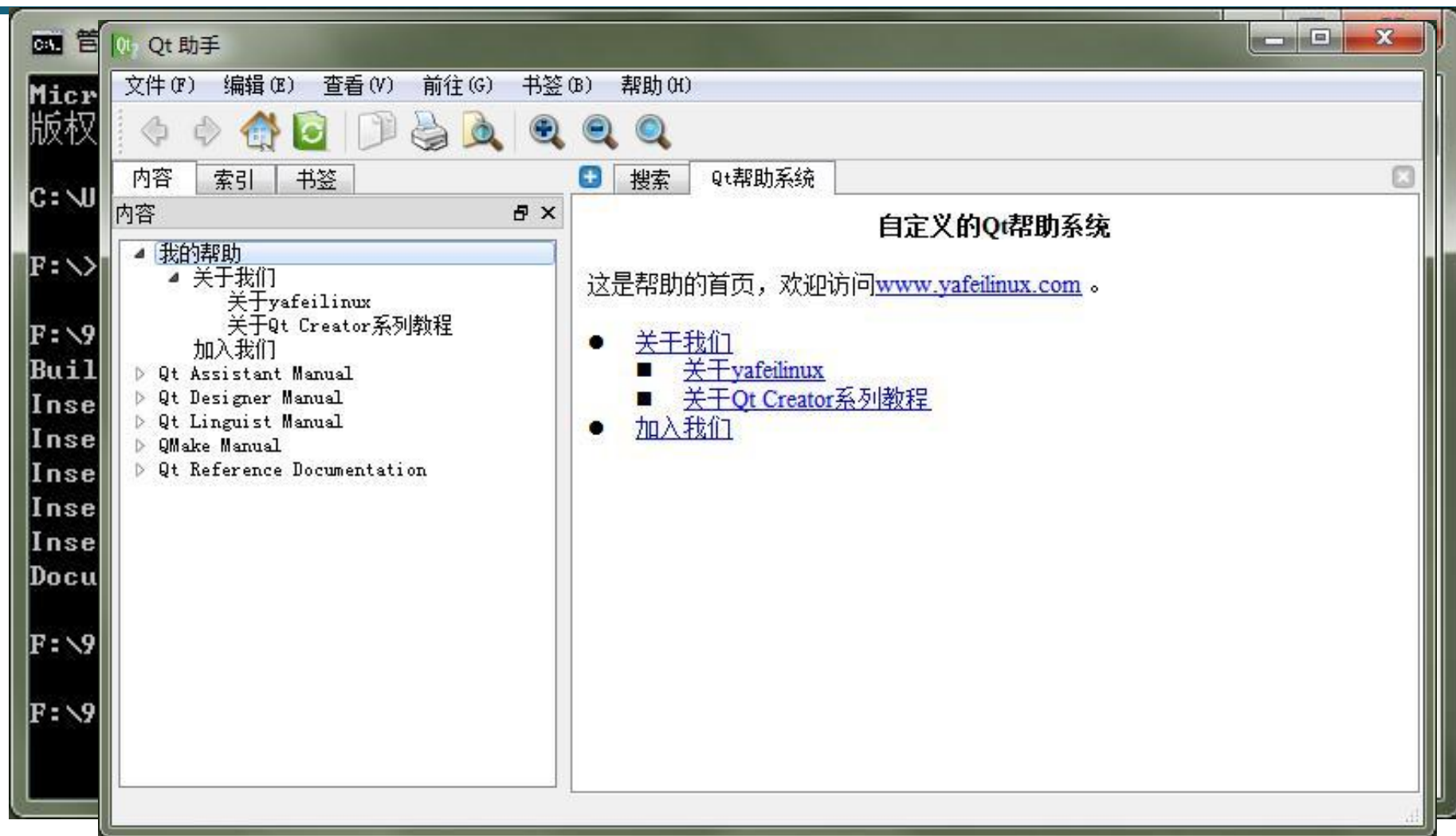
关于 Qt Creator 系列教程

加入我们

要设置过滤器属性；



第三步，生成.qch文件



第四步，创建.qhcp文件

要想使Qt Assistant只显示我们自己的帮助文档的最简单的方法就是生成帮助集合文件即.qhc文件，要生成.qhc文件，首先要创建.qhcp文件。

- 在documentation文件夹中新建文本文件，对其进行编辑，最后另存为“myHelp.qhcp”，注意后缀为.qhcp。
- 这里还要创建一个名为“about.txt”的文本文件，在其中输入一些该帮助的说明信息，作为Qt Assistant的About菜单的显示内容。



```
<?xml version="1.0" encoding="GB2312"?>
```

```
<QHelpCollectionProject version="1.0">
```

```
<assistant>
```

```
<title>我的帮助系统</title>
```

```
<applicationIcon>images/yafeilinux.png</applicationIcon>
```

```
<cacheDirectory>cache/myHelp</cacheDirectory>
```

```
<homePage>qthelp://yafeilinux.myHelp/doc/index.html</homePage>
```

```
<startPage>qthelp://yafeilinux.myHelp/doc/index.html</startPage>
```

```
<aboutMenuText>
```

```
<text>关于该帮助</text>
```

```
</aboutMenuText>
```

```
<aboutDialog>
```

```
<file>./about.txt</file>
```

```
<icon>images/yafeilinux.png</icon>
```

```
</aboutDialog>
```

```
<enableDocumentationManager>>false</enableDocumentationManager>
```

```
<enableAddressBar>>false</enableAddressBar>
```

```
<enableFilterFunctionality>>false</enableFilterFunctionality>
```

```
</assistant>
```

```
<docFiles>
```

```
<generate>
```

```
<file>
```

```
<input>myHelp.qhp</input>
```

```
<output>myHelp.qch</output>
```

```
</file>
```

```
</generate>
```

```
<register>
```

```
<file>myHelp.qch</file>
```

```
</register>
```

```
</docFiles>
```

```
</QHelpCollectionProject>
```

主页homePage和起始页

startPage，这里使用了第

二步中提到的Qt Assistant

的页面的URL，这个URL由

“qthelp://”开始，然后是

在.qhp文件中设置的命名

空间，然后是虚拟文件夹，

最后是具体的HTML文件名。

缓存目录cacheDirectory，是进行全文检索等操作时缓存文件要存放的位置。

因为Qt Assistant可以添加或者删除文档来为多个

应用程序提供帮助，但是这里只是为一个应用程序

提供帮助，并且不希望删除我们的文档，所以

禁用了文档管理器documentation manager，因为

这里的文档集很小，而且只有一个过滤器部分，

所以也关闭了地址栏address bar和过滤器功能，

还关闭了一些没有用的功能。

filter functionality。



第五步，生成.qhc文件



第六步，在程序中启动Qt Assistant

自学内容：

如何在自己的程序中通过点击按钮来启动自定义的帮助系统？



9.3 创建Qt插件

Qt插件 (Qt Plugin) 就是一个共享库 (dll文件)，可以使用它来进行功能的扩展。Qt中提供了两种API来创建插件：

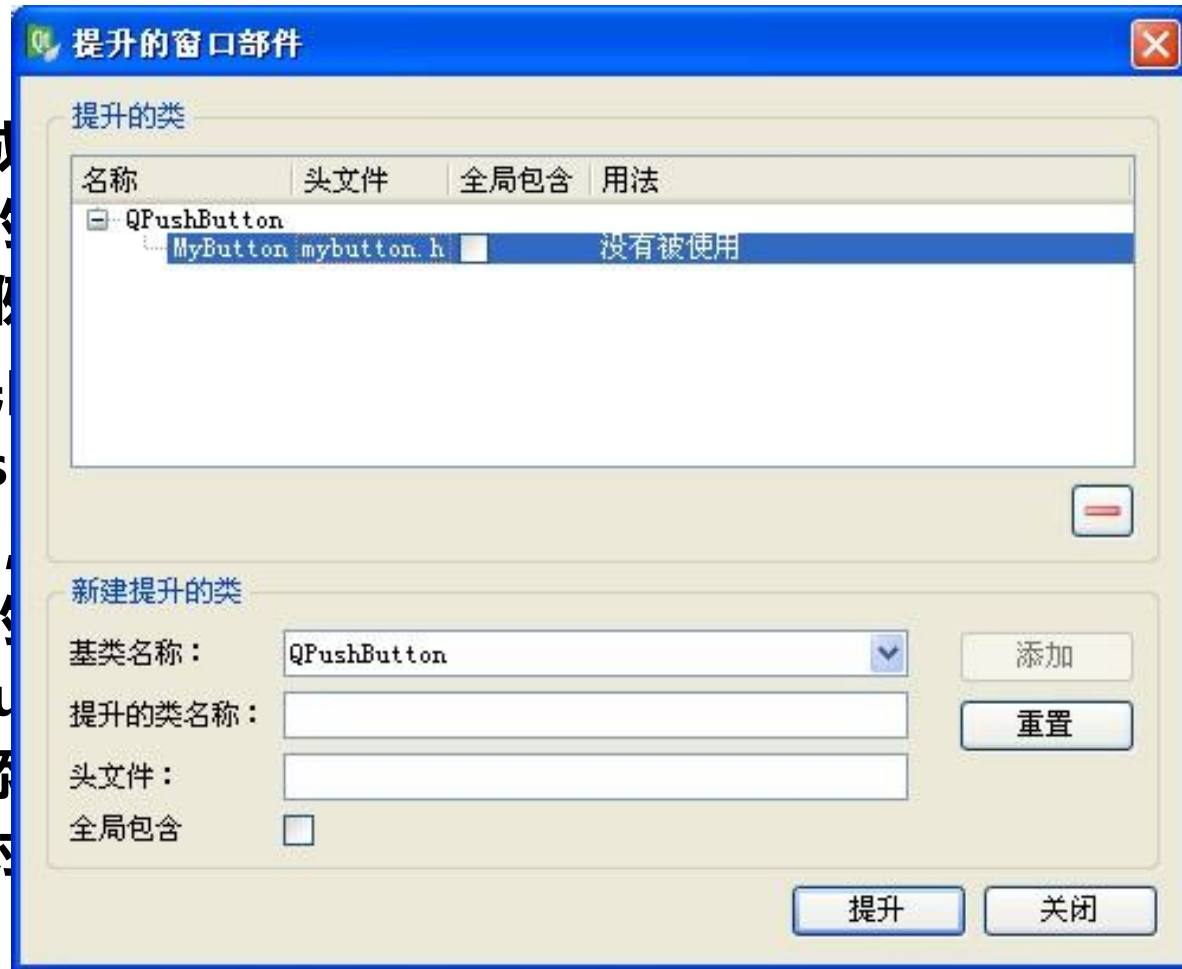
- 用来扩展Qt本身的高级API，如自定义数据库驱动、图片格式、文本编码和自定义风格等；
- 用来扩展Qt应用程序的低级API。

如果要写一个插件来扩展Qt本身，那么可以子类化合适的插件基类，然后重写一些函数并添加一个宏。



在设计模式提升窗口部件

一般的，
过建立成
口部件的
实例。例
先
“QPush
Button，
“提升的
“MyBu
右边的添
钮退出对



但是可以通
用提升窗
义的类的
一个Push
单中选择
为
这时按下
提升”按



创建应用程序插件

当要创建一个插件时，要先创建一个接口，接口就是一个类，它只包含纯虚函数。插件类要继承自该接口。插件类存储在一个共享库中，因此可以在应用程序运行时进行加载。创建一个插件包括以下几步：

- 定义一个插件类，它需要同时继承自QObject类和该插件所提供的功能对应的接口类；
- 使用Q_INTERFACES()宏在Qt的元对象系统中注册该接口；
- 使用Q_PLUGIN_METADATA()宏导出该插件；
- 使用合适的.pro文件构建该插件。

使一个应用程序可以通过插件进行扩展要进行以下几步：

- 定义一组接口（只有纯虚函数的抽象类）；
- 使用Q_DECLARE_INTERFACE()宏在Qt的元对象系统中注册该接口；
- 在应用程序中使用QPluginLoader来加载插件；
- 使用qobject_cast()来测试插件是否实现了给定的接口。



示例

通过创建一个过滤字符串中出现的第一个数字的插件来讲解应用程序的插件的创建过程。

这里需要创建两个项目，一个项目用来生成插件即dll文件；另一个项目是一个测试程序，用来使用插件。因为这两个项目中有共用的文件，所以这里将它们放到一个目录中。



创建插件

- **第一步，创建插件类。**
- **新建空的Qt项目，项目名称为“plugin”，在选择路径时指定到一个新建的“myPlugin”目录中。建立好项目后向其中添加一个C++类，类名为“RegExpPlugin”，基类保持为空。**



第二步，定义插件类。将regexppugin.h文件中的内容更改如下：

```
#ifndef REGEXPPLUGIN_H
#define REGEXPPLUGIN_H
#include <QObject>
#include "regexinterface.h"
class RegExpPlugin : public QObject, RegExpInterface
{
    Q_OBJECT
    Q_PLUGIN_METADATA(IID
        "org.qtexamples.myplugin.RegExpInterface"
        FILE "myplugin.json")
    Q_INTERFACES(RegExpInterface)

public:
    QString regexp(const QString &message);
};
#endif
```



- 为了使这个类作为一个插件，它需要同时继承自QObject和RegExpInterface。RegExpInterface是接口类，它用来指明插件要实现的功能，这个类在regexinterface.h文件中定义，这个文件在后面的测试程序项目中。
- Q_PLUGIN_METADATA()宏用于声明插件的元数据，其中必须指明IID标识符，标识符是一个字符串，必须保证它的唯一性；FILE指定一个JSON格式的插件元数据文件，该参数是可选的，其命名一般使用项目名称即可，内容一般只包含一组大括号。
- 这里还需要使用Q_INTERFACES()宏将这个接口注册到Qt的元对象系统中，告知Qt这个类实现了哪个接口。
- 最后还声明了一个regex()函数，它是在RegExpInterface中定义的一个纯虚函数。这里通过重写它来实现该插件具体的功能，就是将字符串中的第一个数字提取出来并返回。

第三步，导出插件。将regexplugin.cpp文件中的内容更改如下：

```
#include "regexplugin.h"  
#include <QRegExp>  
#include <QtPlugin>
```

```
QString RegExpPlugin::regex(const QString  
    &message)  
{  
    QRegExp rx("\\d+");  
    rx.indexIn(message);  
    QString str = rx.cap(0);  
    return str;  
}
```



第四步，更改项目文件。打开plugin.pro文件，将其内容更改如下：

TEMPLATE = lib

CONFIG += plugin

INCLUDEPATH += ../regexpwindow

HEADERS = regexpplugin.h

SOURCES = regexpplugin.cpp

TARGET = regexpplugin

DESTDIR = ../plugins

- **这里使用了TEMPLATE=lib表明该项目要构建库文件，而不是像以前那样的可执行文件；**
- **使用CONFIG += plugin告知qmake要创建一个插件；**
- **因为项目中使用了regexpwindow目录中的regexpinterface.h文件，所以这里将该目录的路径添加到了INCLUDEPATH中；**
- **TARGET指定了生产的dll文件的名字；**
- **最后使用DESTDIR指定了生成的dll文件所在的目录。**



使用插件扩展应用程序

第一步，新建Qt Widgets应用。项目名称为“regexwindow”，选择路径时仍选择前面建立的myPlugin目录。基类选择QWidget，类名保持“Widget”不变。

建立完成后，向该项目中添加新文件，模板选择C++头文件，名称为“regexinterface.h”。



第二步，定义接口。将regexpinterface.h文件的内容更改如下：

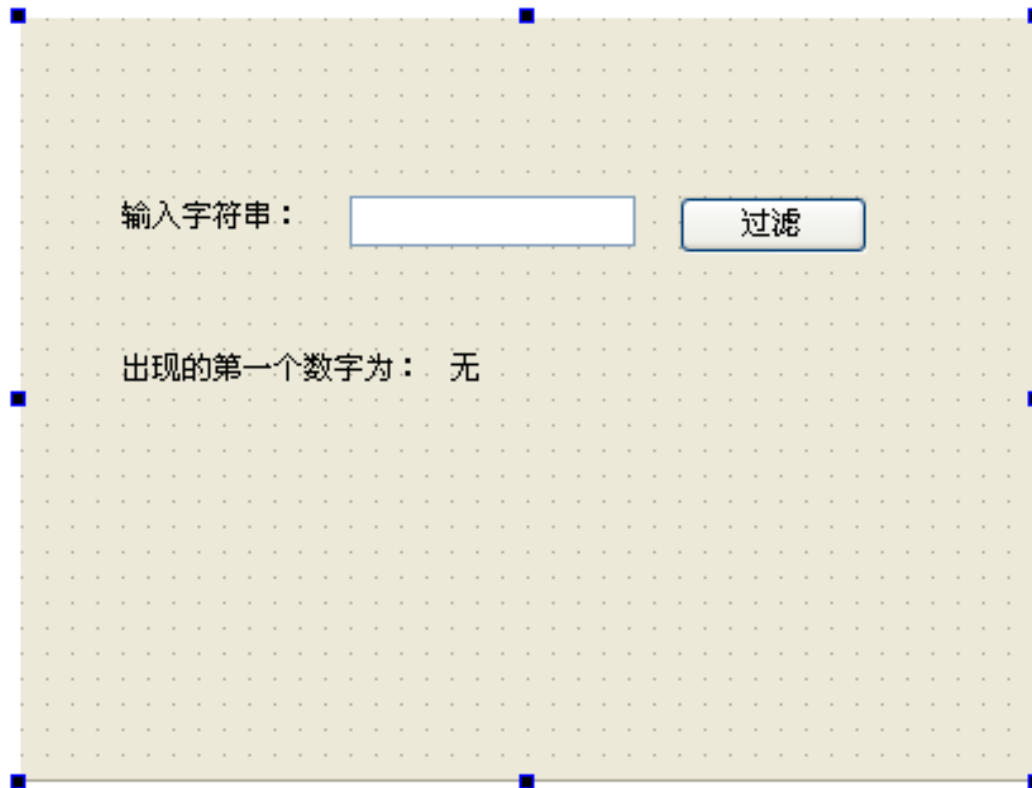
```
#ifndef REGEXPINTERFACE_H  
#define REGEXPINTERFACE_H  
#include <QString>  
class RegExpInterface  
{  
public:  
    virtual ~RegExpInterface() {}  
    virtual QString regexp(const QString &message) = 0;  
};  
Q_DECLARE_INTERFACE(RegExpInterface,  
    "org.qter.Examples.myplugin.RegExpInterface")  
#endif
```

在接口类中定义了插件要实现的函数，比如这里定义了regexp()函数，可以看到在前面的RegExpPlugin类中我们已经实现了该函数。在这个类中只能包含纯虚函数。最后使用Q_DECLARE_INTERFACE()宏在Qt元对象系统中注册了该接口，其中第二个参数就是前面指定的IID。



第三步，加载插件。先点击widget.ui文件进入设计模式，设计的界面。将其中显示“无”字的Label的objectName属性更改为“labelNum”。然后进入widget.h文件，先添加头文件#include “regexpinterface.h”，然后在private部分定义一个接口对象指针，再声明一个加载插件函数：

```
RegExInterface *regexpInterface;  
bool loadPlugin();
```



现在到widget.cpp文件中先添加头文件包含：

```
#include <QPluginLoader>
```

```
#include <QMessageBox>
```

```
#include <QDir>
```

然后在构造函数中调用加载插件函数，如果加载失败则进行警告：

```
if (!loadPlugin()) { // 如果无法加载插件
```

```
    QMessageBox::information(this, "Error",  
                             "Could not load the plugin");
```

```
    ui->lineEdit->setEnabled(false);
```

```
    ui->pushButton->setEnabled(false);
```

```
}
```



下面是加载插件函数的定义：

```
bool Widget::loadPlugin()
{
    QDir pluginsDir("../plugins");
    // 遍历插件目录
    foreach (QString fileName, pluginsDir.entryList(QDir::Files)) {
        QPluginLoader
        pluginLoader(pluginsDir.absoluteFilePath(fileName));
        QObject *plugin = pluginLoader.instance();
        if (plugin) {
            regexplInterface = qobject_cast<RegExplInterface *>(plugin);
            if (regexplInterface)
                return true;
        }
    }
    return false;
}
```

这里使用QDir类指定到存放dll文件的plugins目录，然后遍历该目录，使用QPluginLoader类来加载插件，并使用qobject_cast()来测试插件是否实现了RegExplInterface接口。



最后到设计模式，转到“过滤”按钮的clicked()信号对应的槽，更改如下：

```
void Widget::on_pushButton_clicked()
{
    QString str = regexInterface->regex(ui->lineEdit->text());
    ui->labelNum->setText(str);
}
```

这里就是使用了regexInterface接口的regex()函数来获取lineEdit部件中输入的字符串中第一个出现的数字，然后在labelNum中显示出来。



第四步，运行程序。先构建plugin项目，在编辑模式左侧的项目列表中的plugin目录上点击鼠标右键，选择“构建”。

当构建完
了生成的c

然后运行



创建Qt Designer自定义部件

Qt Designer的基于插件的架构使得它可以使用用户设计的或者第三方提供的自定义部件，就像使用标准的Qt部件一样。

在自定义部件中的所有特性在Qt Designer中都是可用的，这包含了部件属性、信号和槽等。



示例：

第一步，创建项目。新建项目，模板选择“其他项目”分类中的“Qt4 设计师自定义控件”；项目名称为“myDesignerPlugin”；控件类改为“MyDesignerPlugin”，然后在右侧指定图标文件的路径，随意选择一张图片即可，其他选项保持默认；后面步骤全部保持默认，点击下一步直到完成项目的创建。



第二步，更改部件。可以通过修改mydesignerplugin.h和mydesignerplugin.cpp文件来修改部件，就像编写普通的类一样。这里我们进入mydesignerplugin.cpp文件，添加头文件：

```
#include <QPushButton>
```

```
#include <QHBoxLayout>
```

然后在构造函数中添加如下代码：

```
QPushButton *button1 = new QPushButton(this);
```

```
QPushButton *button2 = new QPushButton(this);
```

```
button1->setText("hello");
```

```
button2->setText("Qt!");
```

```
QHBoxLayout *layout = new QHBoxLayout;
```

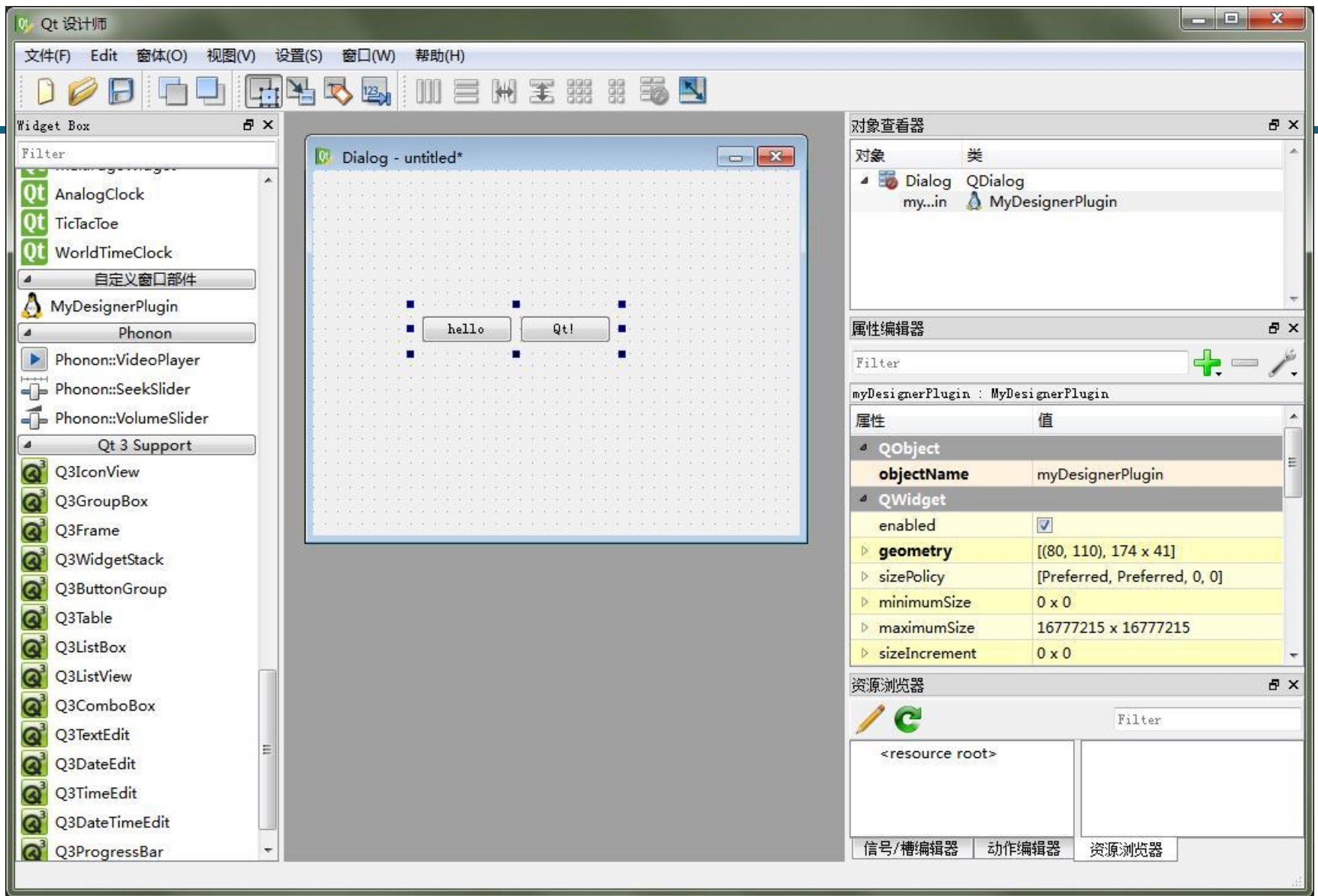
```
layout->addWidget(button1);
```

```
layout->addWidget(button2);
```

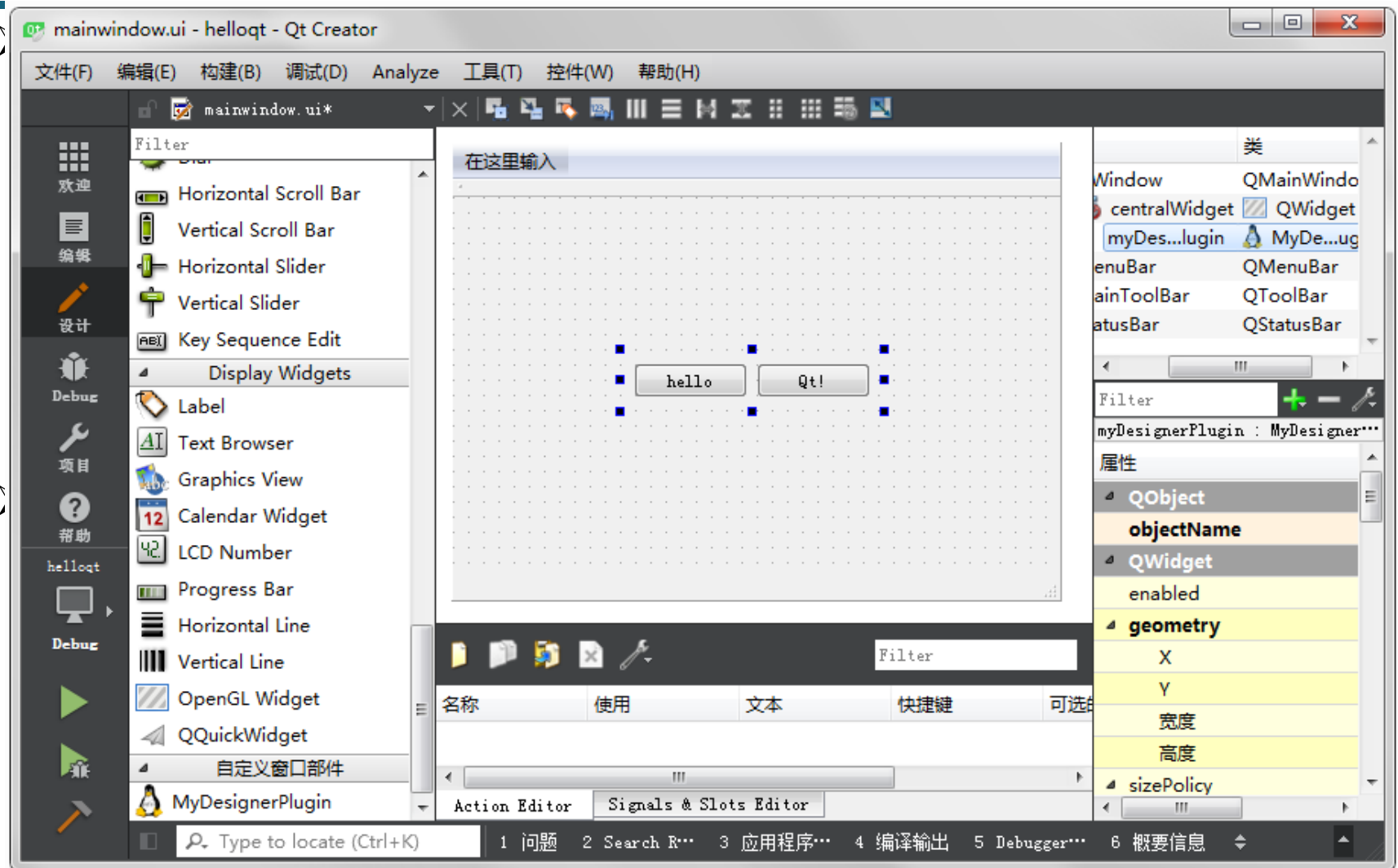
```
setLayout(layout);
```

这里需要在Qt Creator左下角的目标选择器中将构建选项选择为Release，因为只有Release版本的插件在Qt Designer中才可以使用。现在按下Ctrl+Shift+B构建项目，完成后可以看到在项目目录下的build-desktop目录中的release目录中已经生成了相应的dll文件。





说明:



9.4 小结

本章中介绍了Qt中三个比较重要的内容，分别是Qt的国际化、自定义Qt Assistant和自定义Qt插件。这三部分内容看似很复杂，其实很简单，因为这里并没有涉及太多的技术问题，而只是一些流程，以后要使用的时候，按照步骤进行操作就可以了。



- Qt中翻译应用程序的主要步骤？
- Qt 中tr()函数的用法？
- Qt中定制Qt Assistant的主要步骤？
- Qt中创建应用程序插件的主要步骤？

