界面外观

软件学院
马玲 张圣林

一个完善的应用程序不仅应该有实用的功能，还要有一个漂亮的外观，这样才能使应用程序更加友善，更加吸引用户。

作为一个跨平台的UI开发框架，Qt提供了强大而灵活的界面外观设计机制。这一章将学习在Qt中设计应用程序外观的相关知识，在本章开始会对Qt风格QStyle和调色板QPalette进行简单介绍，然后再对Qt样式表（Qt Style Sheets）进行重点讲解，最后还会涉及不规则窗体和透明窗体的实现方法。

主 要 内 容

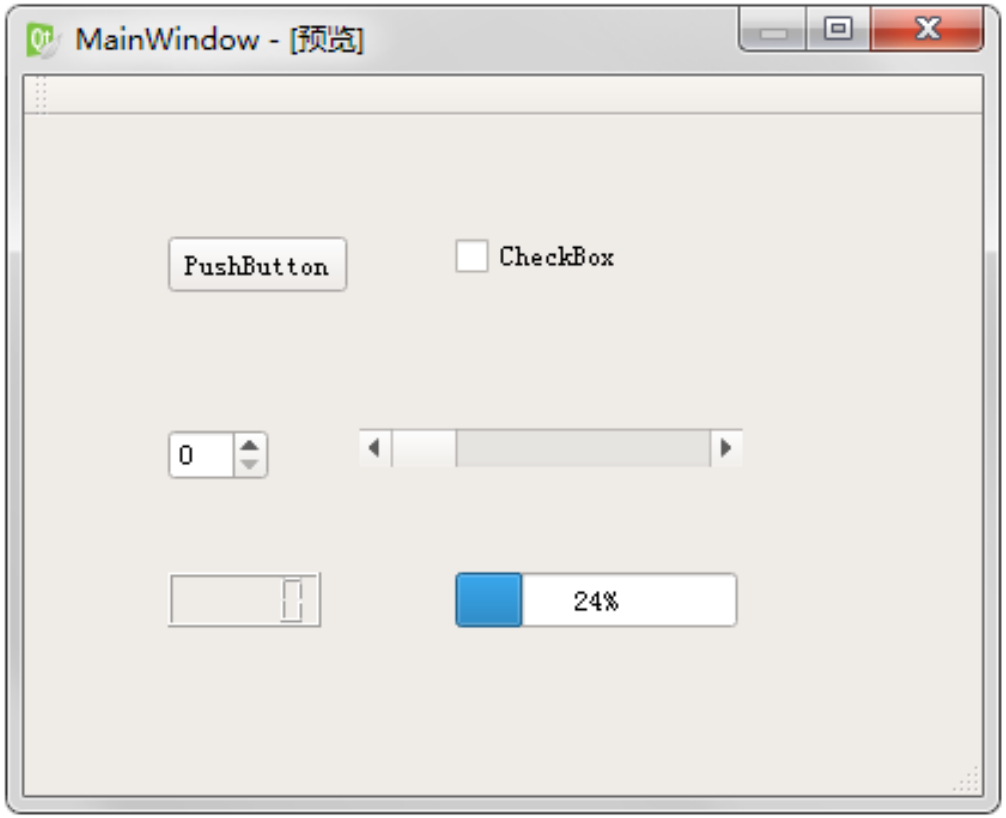
- Qt风格
- Qt样式表
- 特殊效果窗体
- 小结

Qt风格

Qt中的各种风格是一组继承自QStyle的类。QStyle类是一个抽象基类，封装了一个GUI的外观，Qt的内建（built-in）部件使用它来执行几乎所有的绘制工作，以确保它们看起来可以像各个平台上的本地部件一样。QStyleFactory类可以创建一个QStyle对象，首先通过keys()函数获取可用的风格，然后使用create()函数创建一个QStyle对象。一般windows风格和fusion风格是默认可用的，而有些风格只在特定的平台上才有效，例如windowsxp风格、windowsvista风格、gtk风格和macintosh风格。

使用不同风格预览程序

首先进入设计模式，可以先修改界面，然后选择“工具→Form Editor→Preview in”菜单项，这里列出了现在可用的几种风格，选择“Fusion风格”，预览效果如下图所示。也可以使用其他几种风格进行预览。



使用不同风格运行程序

- 如果想使用不同的风格来运行程序，那么只需要调用QApplication的setStyle()函数指定要使用的风格即可。

例如，在main()函数的“QApplication a(argc, argv);”一行代码后添加如下一行代码：

```
a.setStyle(QStyleFactory::create("fusion"));
```

这时运行程序，便会使用Fusion风格。

- 而如果不想整个应用程序都使用相同的风格，那么可以调用部件的setStyle()函数来指定该部件的风格。

调色板

调色板QPalette类包含了部件各种状态的颜色组。一个调色板包含三种状态：激活（Active）、失效（Disabled）和非激活（Inactive）。Qt中的所有部件都包含一个调色板，并且使用各自的调色板来绘制它们自身，这样可以使用户界面更容易配置，也更容易保持一致。调色板中的颜色组包括：

- 激活颜色组QPalette::Active，用于获得键盘焦点的窗口；
- 非激活颜色组QPalette::Inactive，用于其他的窗口；
- 失效颜色组QPalette::Disabled，用于因为一些原因而不可用的部件（不是窗口）。

要改变一个应用程序的调色板，可以先使用QApplication::palette()函数来获取其调色板，然后对其进行更改，最后再使用QApplication::setPalette()函数来使用该调色板。更改了应用程序的调色板，会影响到该程序的所有窗口部件。如果要改变一个部件的调色板，可以调用该部件的palette()和setPalette()函数，这样只会影响该部件及其子部件。

常量	描述
QPalette::Window	一般的背景颜色
QPalette::WindowText	一般的前景颜色
QPalette::Base	主要作为输入部件（如 QLineEdit）的背景色，也可用作 QComboBox 的下拉列表的背景色或者 QToolBar 的手柄颜色，一般是白色或其他浅色
QPalette::AltemateBase	在交替行颜色的视图中作为交替背景色
QPalette::ToolTipBase	作为 QToolTip 和 QWhatsThis 的背景色
QPalette::ToolTipText	作为 QToolTip 和 QWhatsThis 的前景色
QPalette::Text	和 Base 一起使用，作为前景色
QPalette::Button	按钮部件背景色
QPalette::ButtonText	按钮部件前景色
QPalette::BrightText	一种与深色对比度较大的文本颜色，一般用于当 Text 或者 WindowText 的对比度较差时

```
ui->lineEdit->setPalette(palette2);
```

设置调色板颜色时可以使用setColor()函数，这个函数需要指定颜色角色（Color Role）。在QPalette中，颜色角色用来指定该颜色所起的作用，例如是背景颜色或者是文本颜色等，主要的颜色角色如表所示。

Qt样式表

Qt样式表是一个可以自定义部件外观的十分强大的机制。Qt样式表的概念、术语和语法都受到了HTML的层叠样式表（Cascading Style Sheets，CSS）的启发，不过与CSS不同的是，Qt样式表应用于部件的世界。

- Qt样式表概述
- Qt样式表语法
- 自定义部件外观和换肤

Qt样式表概述

- 可以使用QApplication::setStyleSheet()函数将其设置到整个应用程序上；

- 也可以使用QWidget::setStyleSheet()函数将其设置到一个指定的部件（还有它的子部件）上。

- 如果在不同的级别都设置了样式表，那么Qt会使用所有有效的样式表，这被称为样式表的层叠。

使用代码设置样式表

例如：

```
// 设置pushButton的背景为黄色
ui->pushButton->setStyleSheet("background:yellow");
// 设置horizontalSlider的背景为蓝色
ui->horizontalSlider->setStyleSheet("background:blue");
```

这样调用指定部件的setStyleSheet()函数只会对这个部件应用该样式表，如果想对所有的相同部件都使用相同的样式表，那么可以在它们的父部件上设置样式表。比如这里两个部件都在MainWindow上，可以为MainWindow设置样式表：

```
setStyleSheet("QPushButton{background:yellow}QSlider{background:blue}");
```

这样，以后再往主窗口上添加的所有QPushButton部件和QSlider部件的背景色都会改为这里指定的颜色。

在设计模式使用样式表

进入设计模
表”，这时

QPushButton

}

注意光标
箭头，在弹

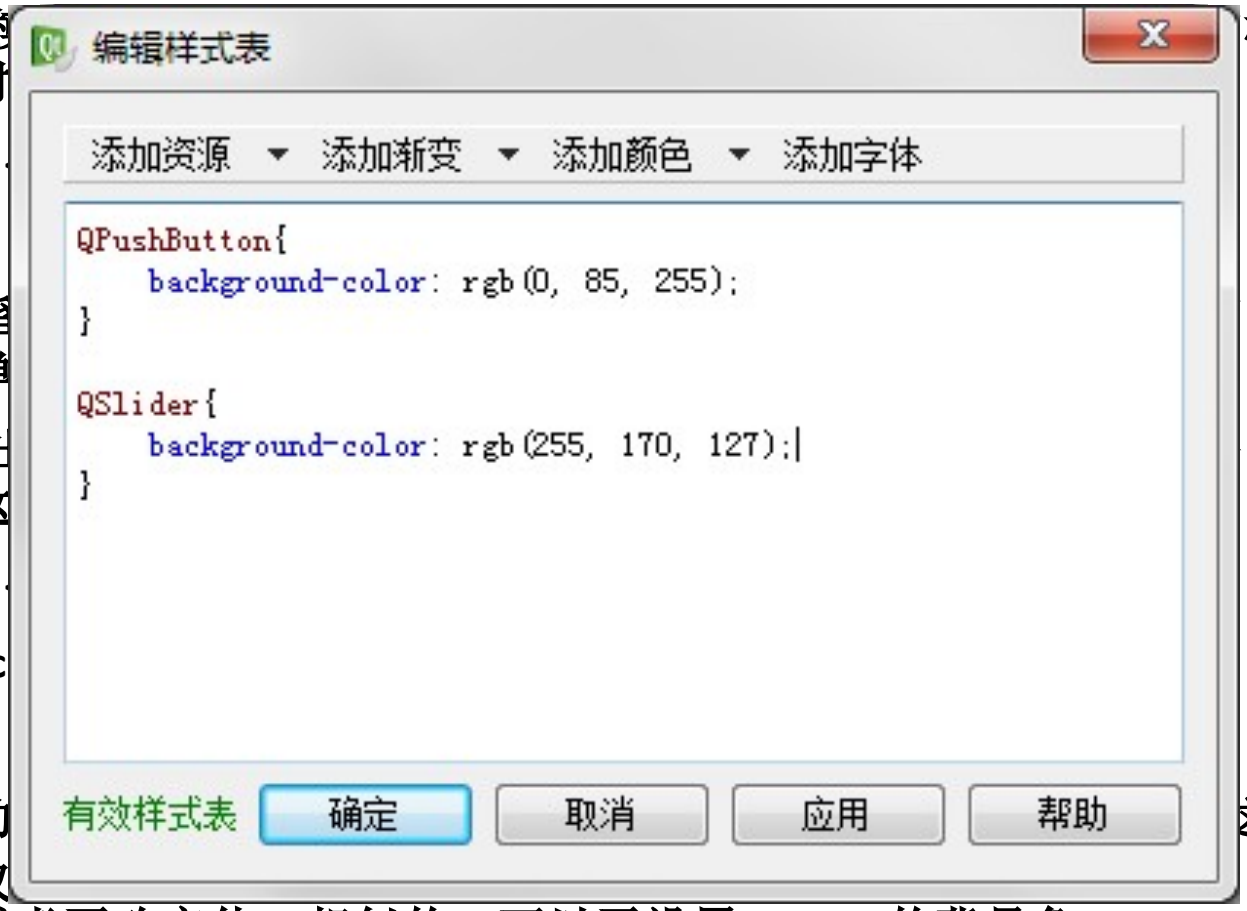
这时会弹出
按钮”，这

QPushButton

background-c

}

根据选择的
样式表不仅



样式

的下拉

“确定

这里设置

使用渐变颜色，或者更改字体。相似的，可以再设置QSlider的背景色。

样式表语法

样式规则：

- 样式表包含了一系列的样式规则，一个样式规则由一个选择器（selector）和一个声明（declaration）组成。选择符指定了受该规则影响的部件；声明指定了这个部件上要设置的属性。例如：

  QPushButton{color:red}

  在这个样式规则中，QPushButton是选择器，{color:red}是声明，而color是属性，red是值。这个规则指定了QPushButton和它的子类应该使用红色作为它们的前景色。
- Qt样式表中一般不区分大小写，例如color、Color、COLOR和COloR表示相同的属性。只有类名，对象名和Qt属性名是区分大小写的。
- 一些选择器可以指定相同的声明，只需要使用逗号隔开，例如：

  QPushButton,QLineEdit,QComboBox{color:red}
- 一个样式规则的声明部分是一些“属性:值”对组成的列表，它们包含在大括号中，使用分号隔开。例如：

  QPushButton{color:red;background-color:white}

选择器类型

Qt样式表支持在CSS2中定义的所有选择器。下表列出了最常用的选择器类型。

选择器	示例	说明
通用选择器	*	匹配所有部件
类型选择器	QPushButton	匹配所有 QPushButton 实例和它的所有子类
属性选择器	QPushButton[flat="false"]	匹配 QPushButton 的属性 flat 为 false 的实例
类选择器	.QPushButton	匹配所有 QPushButton 实例，但不包含它的子类
ID 选择器	QPushButton#okButton	匹配所有 QPushButton 中以 okButton 为对象名的实例
后代选择器	QDialog QPushButton	匹配所有 QPushButton 实例，它们必须是 QDialog 的子孙部件
孩子选择器	QDialog>QPushButton	匹配所有 QPushButton 实例，它们必须是 QDialog 的直接子部件

子控件

对一些复杂的部件修改样式，可能需要访问它们的子控件，例如QComboBox的下拉按钮，还有QSpinBox的向上和向下的箭头等。选择器可以包含子控件来对部件的特定子控件应用规则，例如：

```
QComboBox::drop-down{image:url(dropdown.png)}
```

这样的规则可以改变所有的QComboBox部件的下拉按钮的样式。

伪状态

- 选择器可以包含伪状态来限制规则在部件的指定的状态上应用。伪状态出现在选择器之后，用冒号隔离，例如：

  QPushButton:hover{color:white}

- 这个规则表明当鼠标悬停在一个QPushButton部件上时才被应用。伪状态可以使用感叹号来表示否定，例如要当鼠标没有悬停在一个QRadioButton上时才应用规则，那么这个规则可以写为：

  QRadioButton:!hover{color:red}

- 伪状态还可以多个连用，达到逻辑与效果，例如当鼠标悬停在一个被选中的QCheckBox部件上时才应用规则，那么这个规则可以写为：

  QCheckBox:hover:checked{color:white}

- 如果有需要，也可以使用逗号来表示逻辑或操作，例如：

  QCheckBox:hover,QCheckBox:checked{color:white}

冲突解决

当几个样式规则对相同的属性指定了不同的值时就会产生冲突。例如：

```
QPushButton#okButton { color: gray }

QPushButton { color: red }
```

这样okButton的color属性便产生了冲突。解决这个冲突的原则是：特殊的选择器优先。在这里，因为QPushButton#okButton一般代表一个单一的对象，而不是一个类所有的实例，所以它比QPushButton更特殊，那么这时便会使用第一个规则，okButton的文本颜色为灰色。

相似的，有伪状态比没有伪状态优先。如果两个选择器的特殊性相同，则后面出现的比前面的优先。Qt样式表使用CSS2规范来确定规则的特殊性。

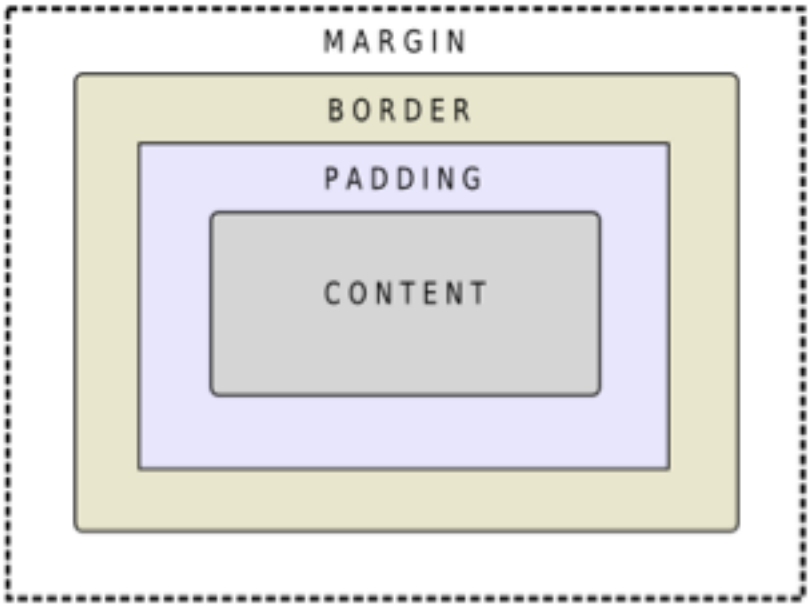
层叠

样式表可以被设置在QApplication上，或者父部件上，或者子部件上。部件有效的样式表是通过部件祖先的样式表和QApplication上的样式表合并得到的。

当发生冲突时，部件自己的样式表优先于任何继承的样式表，同样，父部件的样式表优先于祖先的样式表。

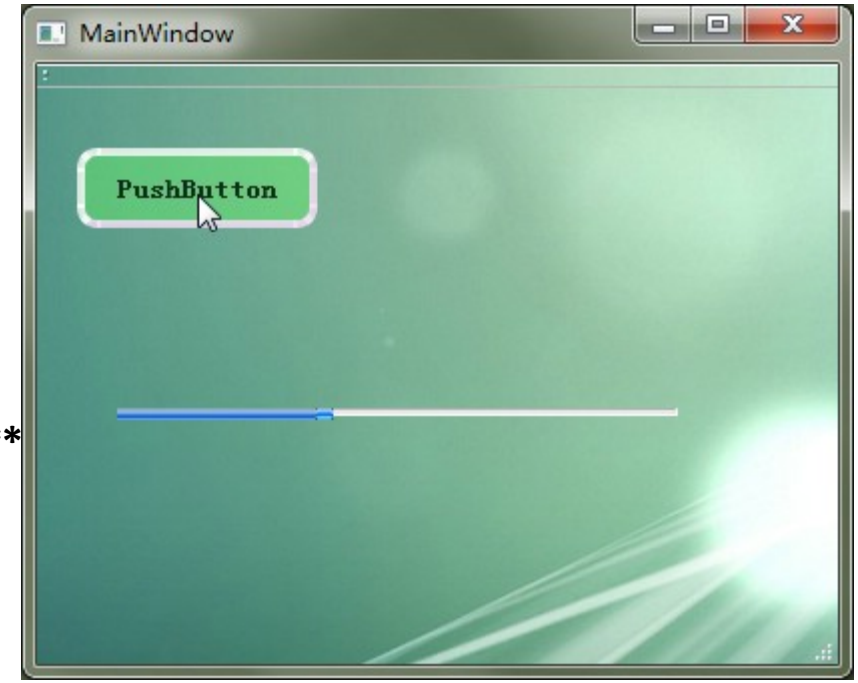
自定义部件外观

当使用样式表时，每一个部件都被看做是拥有四个同心矩形的盒子，如下图所示。这四个矩形分别是内容（content）、填衬（padding）、边框（border）和边距（margin）。边距、边框宽度和填衬等属性的默认值都是0，这样四个矩形恰好重合。



```
/****************主界面背景******************/
QMainWindow{
/*背景图片*/
background-image: url(:/image/beijing01.png);
}
/****************按钮部件******************/
QPushButton{
/*背景色*/
background-color: rgba(100, 225, 100, 30);
/*边框样式*/
border-style: outset;
/*边框宽度为4像素*/
border-width: 4px;
/*边框圆角半径*/
border-radius: 10px;
/*边框颜色*/
border-color: rgba(255, 225, 255, 30);
/*字体*/
font: bold 14px;
/*字体颜色*/
color:rgba(0, 0, 0, 100);
/*填衬*/
padding: 6px;
}
```

```
/*鼠标悬停在按钮上时*/
QPushButton:hover{
background-color:rgba(100,255,100, 100);
border-color: rgba(255, 225, 255, 200);
color:rgba(0, 0, 0, 200);
}
/*按钮被按下时*/
QPushButton:pressed {
background-color:rgba(100,255,100, 200);
border-color: rgba(255, 225, 255, 30);
border-style: inset;
color:rgba(0, 0, 0, 100);
}
/*****************滑块部件*********************
/*水平滑块的手柄*/
QSlider::handle:horizontal {
image: url(:/image/sliderHandle.png);
}
/*水平滑块手柄以前的部分*/
QSlider::sub-page:horizontal {
/*边框图片*/
border-image: url(:/image/slider.png);
}
```



实现换肤功能

Qt样式表可以存放在一个以.qss为后缀的文件中，这样我们就可以在程序中调用不同的.qss文件来实现换肤的功能。

在换肤按钮中这样设置：

```
void MainWindow::on_pushButton_clicked(){
    QFile file(":/qss/my1.qss");
    file.open(QFile::ReadOnly);
    QString styleSheet = tr(file.readAll());
    qApp->setStyleSheet(styleSheet);
}
```

例如通过下面的代码使用样式表：

```
QFile file(":/qss/my.qss");
// 只读方式打开该文件
file.open(QFile::ReadOnly);
// 读取文件全部内容，使用tr()函数将其转换为QString类型
QString styleSheet = tr(file.readAll());
//为QApplication程序设置样式表
qApp->setStyleSheet(styleSheet);
```

现在可以运行程序了，当大家按下按钮后，便会更改界面的外观，这样就实现了换肤功能。

这里读取了Qt样式表文件中的内容，然后为应用程序设置了样式表。

特殊效果窗体

- 不规则窗体
- 透明窗体

不规则窗体

Qt中提供了部件遮罩（mask）来实现不规则窗体。例如：

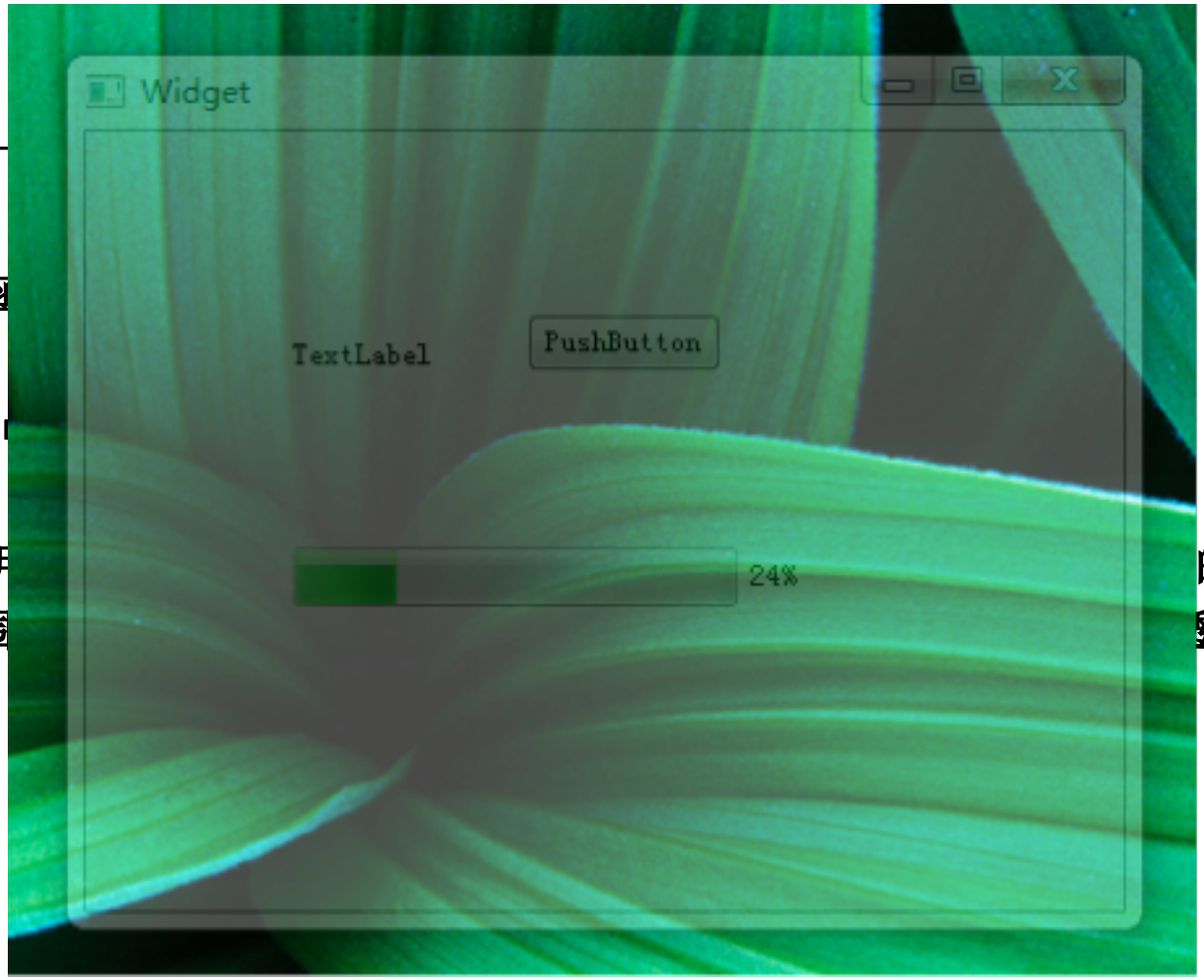
- 先在构造函数中添加如下代码：

```
QPixmap pix;
pix.load(":/image/yafeilinux.png"); // 加载图片
resize(pix.size()); // 设置窗口大小为图片大小
setMask(pix.mask()); // 为窗口设置遮罩
```

- 然后在paintEvent()函数中将图片绘制在窗口上：

```
void Widget::paintEvent(QPaintEvent *)
{
    QPainter painter(this);
    // 从窗口左上角开始绘制图片
    painter.drawPixmap(0,0,QPixmap(":/image/yafeilinux.png"));
}
```

透明窗体



方式一

构造函

setWi

使用 的参数取值范围 透明。

方式二：使用setAttribute()函数。例如：

在构造函

setWindov

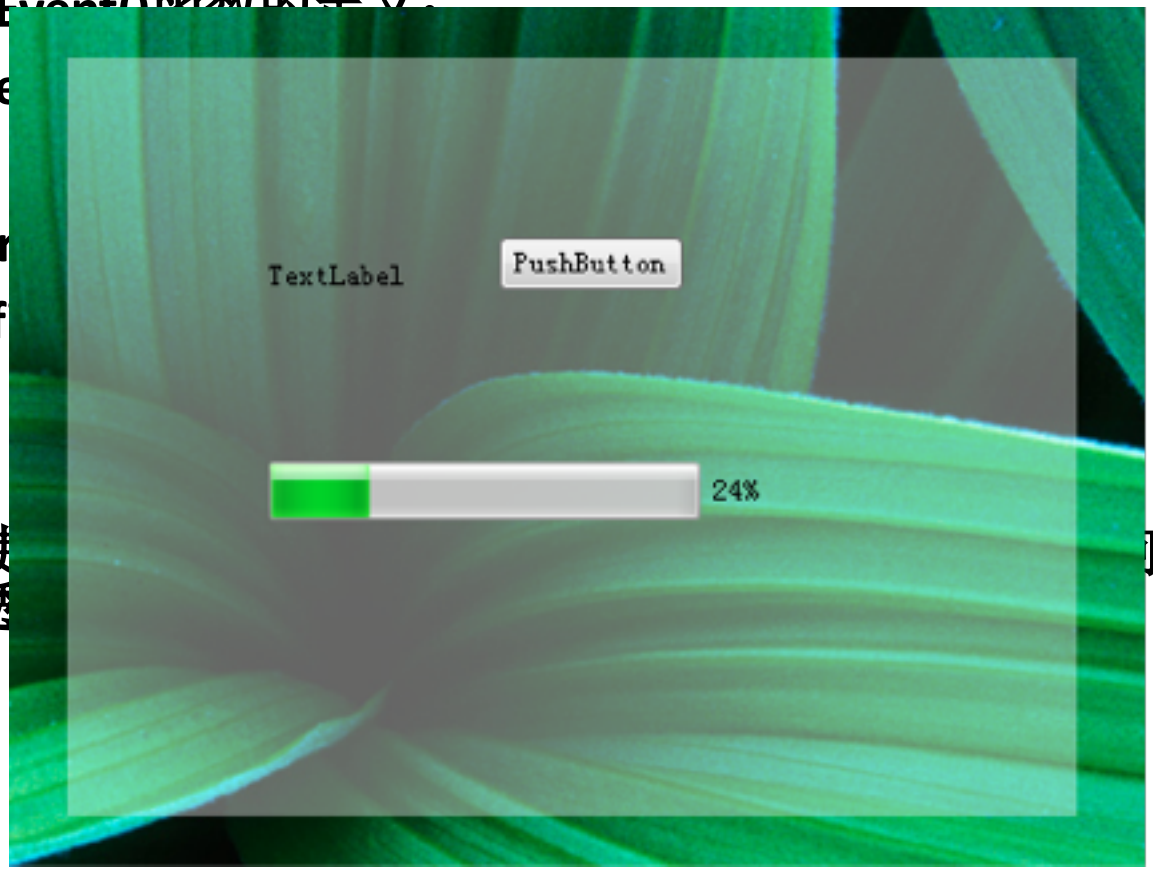
setAttribut

这里使用 属性，它
可以使窗 要使用
setWind 现透明效果。

方式三：在方式二的基础上修改重绘事件。例如：

进行paintEvent()函数的定义。

```
void Widge
{
    QPainter
    painter.f
}
```



这里先使 边框。然后
使用半透

小结

本章要掌握最基本的更改部件样式的方法。通过综合使用本章讲到的这些知识，应该可以实现一些简单的界面效果。重点掌握Qt样式表的用法。