自定义外观

三种方法来重新定义Qt内置窗口部件的外观：

* 子类化个别的窗口部件类，并且重新实现它的绘制和鼠标事件处理器。
* 子类化QStyle或者一个预定义的风格，例如QWindowStyle。
* 使用Qt样式表。

使用Qt样式表

使用样式表

int main(int argc, char \*argv[])

{

QApplication a(argc, argv);

QApplication::setStyle(QStyleFactory::create("Fushion"));

QFile file(":/qss/candy.qss");

file.open(QFile::ReadOnly);

a.setStyleSheet(file.readAll());

CandyDialog w;

w.show();

return a.exec();

}

样式表作用于上层的当前激活的QStyle上。

使用样式表与修改窗口部件的调色板更为可靠。因为QPalette中的实体（Base、Button等）在不同样式中的用法是不一样的。此外，有些样式使用硬编码的图像呈现某个元素，从而跳过了调色板。样式表保证了无论激活的是哪种样式，都可以使用指定的颜色。

QApplication::setStyleSheet()为整个应用程序设置一个样式表：

QApplication::setStyleSheet(“QLineEdit { background-color: yellow; }”);

QWidget::setStyleSheet()设置窗口部件及其子窗口部件的样式表：

dialog->setStyleSheet(“QLineEdit { background-color: yellow; }”);

直接在QLineEdit上设置样式表，可以忽略QLineEdit选择器及大括号：

lineEdit->setStyleSheet(“background-color: yellow;”);

样式表的规则

样式规则通常是组合的。例如：

QCheckBox, QComboBox, QLineEdit {

color: #050505;

/\*color: rgb(0, 88, 152);\*/

/\*color: rgba(97%, 87%, 9%, 50%);\*/

/\*color: palette(Base);\*/

/\*color: qlineargradient(x1: 0, y1: 0, x2: 1, y2: 1, stop: 0 white, stop: 0.4 gray, stop: 1 green);\*/

background-color: yellow;

}

样式表选择器语法

|  |  |  |
| --- | --- | --- |
| **选择器** | **实例** | **可以匹配的窗口部件** |
| **全局对象** | \* | 任意窗口部件 |
| **类型** | QDial | 给定类的实例，包含子类 |
| **类** | .QDial | 给定类的实例，不包含子类 |
| **标识** | QDial # ageDial | 给定对象名称的窗口部件 |
| **Qt属性** | QDial[y = “0”] | 为某些属性赋值的窗口部件 |
| **子对象** | QFrame > QDial | 给定窗口部件的直接子窗口部件 |
| **子孙对象** | QFrame QDial | 给定窗口部件的子孙窗口部件 |

例如：为了选择所有称作“okButton”的QPushButton，它们的x和y属性为0，名为“frame”的QFrame直接子对象，可以这样写：

QFrame#frame > QPushButton[x=”0’][y=”0”]#okButton { … }

Qt可以动态的设置一个不存在的属性的值，创建一个自定义属性。例如：

nameLineEdit->setProperty(“mandatoryField”, true);

….

\*[ mandatoryField=”true”] {…}

样式表辅助控制器

|  |  |
| --- | --- |
| **辅助控制器** | **说明** |
| **::indicator** | 复选框、单选钮、可选菜单项或可选群组框的指示器 |
| **::menu-indicator** | 按钮的菜单指示器 |
| **::item** | 菜单、菜单栏或状态栏项 |
| **::up-button** | 微调框或滚动条的向上按钮 |
| **::down-button** | 微调框或滚动条的向下按钮 |
| **::up-arrow** | 微调框、滚动条或标题视图的向上按钮 |
| **::down-arrow** | 微调框、滚动条、标题视图或组合框的向上按钮 |
| **::drop-down** | 组合框的下拉箭头 |
| **::title** | 群组框的标题 |

辅助控制器由两个冒号指定。

样式表的状态

|  |  |
| --- | --- |
| **状态** | **说明** |
| **:disabled** | 禁用窗口部件 |
| **:enabled** | 启用窗口部件 |
| **:focus** | 窗口部件有输入焦点 |
| **:hover** | 鼠标在窗口部件上悬停 |
| **:pressed** | 鼠标按键点击窗口部件 |
| **:checked** | 按钮已被选中 |
| **:unchecked** | 按钮未被选中 |
| **:indeterminate** | 按钮被部分选中 |
| **:open** | 窗口部件位于打开或扩展状态 |
| **:closed** | 窗口部件位于关闭或销毁状态 |
| **:on** | 窗口部件的状态是“on” |
| **:off** | 窗口部件的状态是“off” |

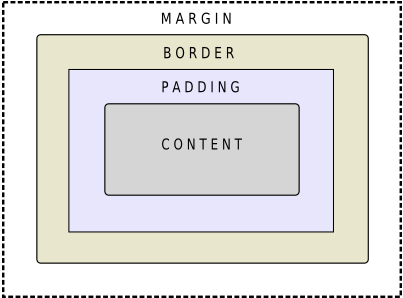
状态由单个的冒号指定。可以一个接一个列出几个状态，它们彼此用冒号隔开，说明窗口部件满足所有的状态时，规则才被使用。

使用多个选择器，用逗号隔开，可以在任何一个状态为true的情况下使用规则。

逻辑否可以用感叹号表示。

CSS盒子模型

此模型可以指定4个影响布局的矩形，从而绘制一个自定义的窗口部件。



* contents 是最里面的矩形。它是绘制窗口部件内容（例如文字和图片）的地方。
* padding包围content。它负责由padding属性指定填充操作。
* border包围padding。它为边界预留空间。
* margin是最外边的矩形。它包围border，负责任何指定的边缘空白区域。

对于没有padding、border和margin的普通窗口部件，这4个矩形重合在一起。

样式表属性

当一个属性被具有同一选择器的几个规则同时设置时，只有最后一个规则起作用。

background-image

QLineEdit {

background-image: url(:/images/yellow-bg.png);

background-position: top right;

background-repeat: repeat-y;

}

background-image指定一个默认图片作为背景。默认情况下，背景图片从窗口的左上角（不包含使用margin指定的边缘区域）开始，并且向水平和竖直方向重复填充整个窗口部件。可以通过使用background-position和background-repeat属性设置。

border

QLineEdit {

border: 2px groove gray;

border-radius: 10px;

padding: 2px 4px

}

border属性指定一个灰色、2像素宽的“凹槽”边框，可以分别设置为border-width、border-style以及border-color。

border-radius把边框的角设为圆角，这里半径为10像素。

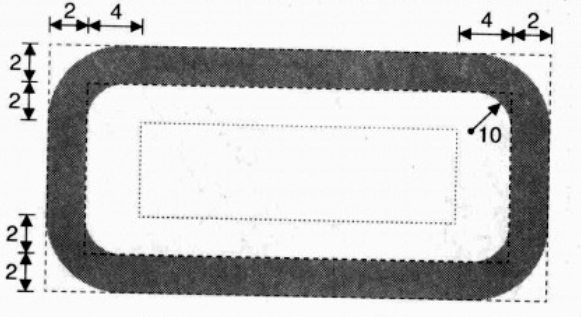
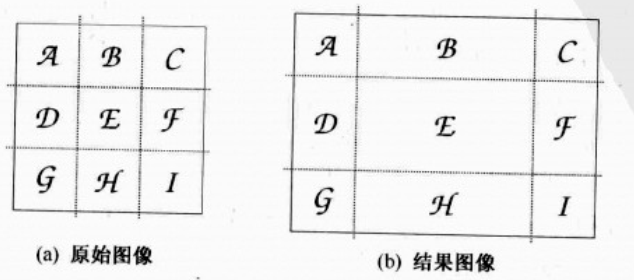


图 1QLineEdit 的结构

border-image

与使用background-image定义的背景图像不同，边界图被分割成3 x 3的小格，如下图所示。当填充窗口部件背景时，4个角（A、C、G和I）保持不变，其他5个格子被拉伸或平铺，填充可用空间。



使用border-image属性可以指定各个边界图，它要求指定一个图形文件名和定义9个格子的4条切线。切线用其到上、右、下和左边缘的距离定义。例如：

border-image: url(border.png) 4 8 12 16

当使用边界图时，必须显示的设置border-width属性。一般情况下，border-width应该与切线的位置一致；否则，为了与border-width相符合，角上的格子将被拉伸或缩短。例如，对于上述border-image：

border-width: 4px 8px 12px 16px

例如，绘制一个QPushButton：

QPushButton {

color: white;

font: bold 10pt;

border-image: url(:/images/button.png) 16;

border-width: 16px;

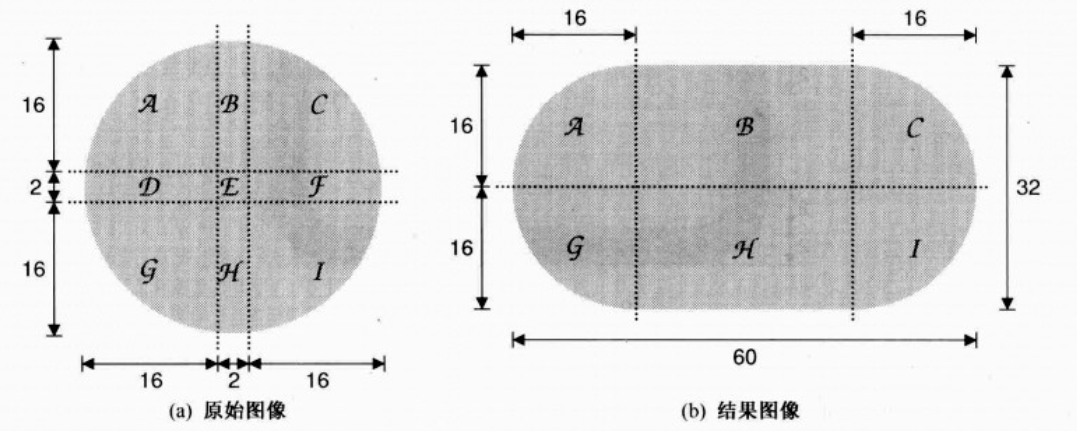
padding: -16px 0px;

min-height: 32px;

min-width: 60px;

}

QPushButton边界图的4条切线位于距离34 X 34的边界图（button.png文件大小为34 X 34大小）16像素的位置，如下图（a）所示：



在图（b）中，对应于D、E和F的边界图的格子被丢弃，因为缩小的按钮不足以显示，会水平拉伸格子B和H，以便占用余下的宽度。

color & background-color

颜色可以由名称、#RRGGBB格式的HTML样式的字符串、RGB或RGBA值、调色板、渐变指定。使用名称时，可以使用任何能被QColor::setNameColor()函数识别的名字。

font

QLable { font: 9pt; }

padding

QLineEdit {

padding: 0px 15px 0px 0px

}

padding属性允许我们指定窗口部件的上边、右边、下边和左边的填充空间。填充的位置在QLineEdit的文字和窗体之间。CSS同时定义了padding-top、padding-right、padding-bottom、padding-left。

subcontrol-origin & subcontrol-position

QComboBox::drop-down:!editable {

subcontrol-origin: padding;

subcontrol-position: center right;

}

使用subcontrol-origin和subcontrol-position属性，下拉箭头竖直居中，位于只读组合框的填充矩形的右侧。

子类化QStyle

QStyle体系能够通过子类化QStyle或者以现存的样式来开发新的自定义外观。在现存的样式上，我们可以做最少的修改，或者可以从无到有的创建一种完全自定义的形式。

QStyle API包含绘制图形元素【drawPrimitive()、drawControl()、drawComplexControl()等】的函数，以及样式查询函数【pixelMetrics()、styleHint()、hitTest()等】。QStyle成员函数典型的带有QStyleOption对象，它包含绘制窗口部件的通用信息（例如其调色板）以及特有信息（例如按钮的文字）。函数还包含一个可选的QWidget指针，以应对QStyleOption不能提供全部所需信息的情况。

通常使用QApplication::setStyle()设置整个应用程序的样式，使用QWidget::setStyle()为个别窗口部件设置样式。

QWidget::style()函数返回绘制窗口部件的合适的样式。

QStyleOption::initFrom()函数初始化用于显示窗口部件的基本成员变量，例如rect、state（是否可用、是否获得焦点等），以及调色板。具有特性变量的QStyleOptionButton等的成员变量必须被手动的初始化，否则将使用默认值。

QStyle API

自定义样式也可以被编译成插件，在Qt设计师中使用，使用该样式预览窗体。

polish & unpolish

polish()和unpolish()函数会在安装和卸载样式的时候得到调用。它们可以让我们对窗口部件或调色板做适当修改。

自定义外观的颜色配置有两种设置方式：

* 忽略窗口部件的QPalette，使用我们喜欢的颜色进行绘制；
* 重新实现polish(QPalette& )调整应用程序或窗口部件的调色板，然后使用该调色板。

第二种方法更灵活，因为可以在子类中通过重新实现polish()覆盖原有的配色方案。

在窗口部件创建完之后并且在第一次显示之前，会使用当前的样式来调用polish()函数，然后，只有在当前样式动态改变时被调用。

当动态改变样式时，同样会调用unpolish()。unpolish()的目的是取消polish()的作用，这样窗口部件可以被一个新的样式抛光。

polish(QWidget\*)的一般用法是把样式子类作为窗口部件上的事件过滤器。

styleHint

styleHint()函数返回一些关于样式外观的提示。例如，对于SH\_DialogButtonLayout返回MacLayout，表示我们想要QDialogButtonBox依从Mac OS X的风格。它的返回类型是int，对于不能用整数表示的少数的样式提示，styleHint()提供一个指向QStyleHintReturn对象的指针。

pixelMetric

pixelMetric()函数返回一个像素值，用于用户界面元素中。通过重新实现这一功能，既影响内置Qt窗口部件的绘制，也影响其尺寸。

子类化窗口部件类

通过对一个已存在的Qt窗口部件进行子类化或者直接对QWidget进行子类化，就可以创建自定义窗口部件。也可以把自定义的窗口部件集成到Qt设计师中。

Question: Redhat系统中初次构建Qt5程序，出现can not find -lgl错误，因为Qt5默认安装了opengl，所以系统中需要安装opengl，使用root用户执行语句yum install mesa-libGL-devel即可解决问题。