

嗯 Qt 样式使用说明

一、声明：

1. 文档说明：

- **学习 Qt 样式表前提：**

如果你了解 CSS，那么我相信这对你来说会很容易理解并应用于实际界面美化，它与 CSS 有些相似之处，当然如果同时也了解 Qt，那么你会很快学会如何运用 Qt 样式为程序设计出漂亮的界面。

- **阅读本说明文档：**

首先一、阅读声明，二、名词解释，三、语法简介，六、附属例题解释，每次实验并对照着五、样式表参照表，体会并学习如何运用样式表里关键词。

- **样式表参照表之间的关系很复杂：**

很多需要重复描述的选择器、属性、值，但由于本文档不能跳转，所以要自己去查找，按照它语法关系层层迭代定位表格，并按照字母表顺序定位要查找内容的大概范围。这里的表格以 Qt help 文档为准。

- **暂不介绍：**

为了尽快完成本文档的初步可使用的目的，很多在本程序中暂时用不到的窗口部件暂未解释介绍，窗口部件介绍表格中会提到“暂不介绍”，但是辅助控制器，属性，值都是完全展示在表格中。

- **附加的例题格式说明：**

例子是经过实验得出的结论，用 C 语言的注释方式添加到相应的样式代码后面，当然你的文件（.CSS）同样满足这个注释方式。

```
/* 注释内容 */
```

```
/*imagesForExample: example_for_XXX000.png */实例图片
```

- **如何做实验：**

Qt 源码 bin 目录下的 designer.exe, 运行/拖入控件/右键/改变样式表/应用；

Qt Creator 也可以。

- **遇到问题请尽快联系作者**

2. 背景介绍：

Qt 为图形界面应用程序提供一个完整的 C++ 应用程序开发框架。

Qt 的样式表主要是受到 CSS 的启发，通过调用 `QWidget::setStyleSheet()` 或 `QApplication::setStyleSheet()`，你可以为一个独立的子部件、整个窗口，甚至是整个应用程序指定一个样式表。样式表由影响窗口部件绘制的样式规则组成。这些规则都是普通文本。由于在运行时会解析样式表，所以可以通过定制样式表的方式来尝试设计不同的 Qt 应用程序。

二、名词解释：

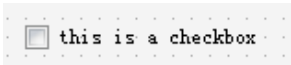
1. 选择器（selector）

意思是：选择特定的类，一般为一个可以定制样式表的 Qt 类，所有可以作为选择器的 Qt 类都在五、样式参照表：2. 可以应用样式表的窗口部件表=选择器中列出，选择器的格式参照五、样式参照表：1 样式选择器类型表。

所谓的选择器可以理解为 CSS 中的选择器，他指定了一类部件进行设计。

2. 辅助控制器（sub-control）

辅助控制器 一词是相对于选择器存在的，可以理解为我们选择了一个部件，例如一个

QCheckBox,  这个部件它分为两个部分，文本部分和可以点击的小窗口的部分。而这个可点击的小窗口部分我们要单独的设置，就要再次分离出来，就需要::indicator(QCheckBox 有这个辅助控制器)来设置，如下例题：

```
QCheckBox::indicator{
    width:20px;
    height:20px;
}
/*说明是在 QCheckBox 中的指示符（indicator）宽为 20px，height 为 20px。*/
```

辅助控制器是用 :: 双冒号进行指定。

如果没有::indicator 那么我们这个小例子将是对整个 QCheckBox 设置的，使用了辅助控制器的时候就自动分离出这个小窗口，对小窗口进行设置。

不同的选择器有不同的辅助控制器，具体可查看五、样式表参照表：3. 辅助控制器列表中详细介绍相应的辅助控制器在不同的类中应用，详细说明在类中的什么位置。

3. 状态（pseudo-states）

除了辅助控制器对一个部件的分离，样式表还可以根据窗口部件的各个状态来设置窗口。例如 hover 表示鼠标划过时的状态，例子如下：

```
QCheckBox:hover{
    color: red;
}
/*例子说明只有当鼠标滑过复选框文本时变为 red*/
```

状态是用 : 冒号进行区分每一个状态。

更详细的状态列表在五、样式表参照表：3 状态列表中查找

状态可以多个一起使用，也可以和辅助控制器一起使用，这样设置窗口部件的时候分的会更加详细。

如下小例子：

```
QCheckBox:hover,QCheckBox:checked{
```

```
        color:red;
    }
    QPushButton::hover{
        color:red;
    }
}
```

4. 属性

它是一个窗口部件所固有的特征、性质，每一个窗口部件都会有属于他们自己的属性。如前面做的小例子中我们一直未曾提过 `color,width,height` 等。组合多个属性同时使用设计出多种效果。**五、样式表参照表：4 属性列表**查找有更多的详细介绍。

5. 值

是属性：后面跟随的一组数字，颜色或者是一个 `bool` 类型等这些我们称它为值，这些值决定了窗口部件的最终展示效果。

查看值的表达方式

五、样式表参照表：4 值列表

6.逻辑否 (!)

有时候我们在设置某种状态的属性时，希望同时在某些非(!)的状态下设置，这个时候我们就要用(!)来选择某种状态，比如!checked、!has-children（没有子目录）等等。

7. 盒模型 (The Box Model)

这个模式指定了 4 个影响布局的矩形，从而绘制一个自定义的窗口部件。

1.Content rectangle 是最里面的矩形，它绘制窗口部件内容（如文字，图片）的地方。

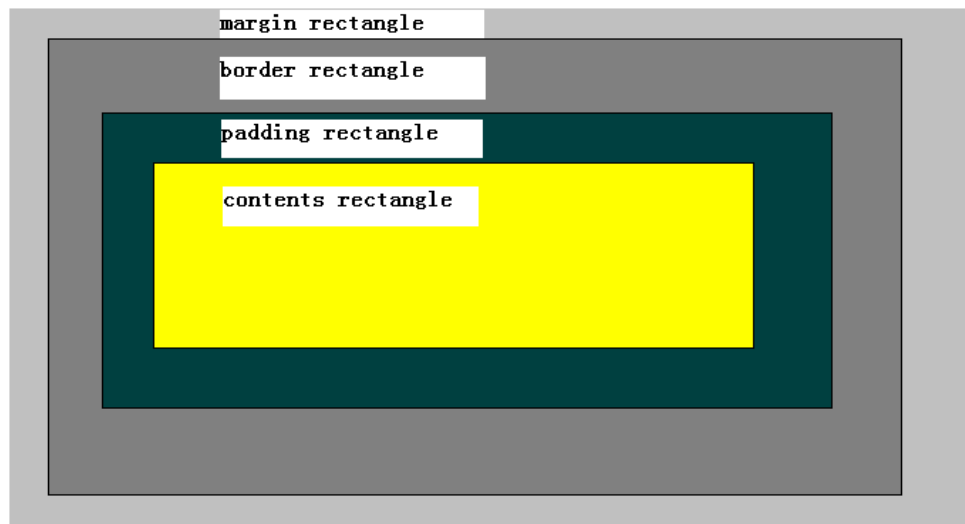
2.padding rectangle 包围 content rectangle。它负责由 **padding** 属性指定填充操作。主要是窗口部件内容与边缘线(border)之间的空隙，它可以用 **top, right, bottom** 和 **left** 设置它的大小。

3.border rectangle 包围 padding rectangle。它为边界预留空间。可以认为是窗口的外框线。下面讲的分割图形的方法中把 border 当做是一个区域来理解的。参考**四、高级应用：九宫格分割法**

4.margin rectangle 最外面的矩形，他包围 border rectangle，负责指定的边缘空白区域，主要是负责与其他的窗口部件的距离。

如果没有指定他们四个，则默认是四个重合在一起的。

如图：



8.角弧度

窗口部件四个角弧度。`radius` 设置角的弧度，如 `border-radius:4px;` 角的弧度是 4px。

9.背景色和前景色

部件的前景色用于绘制窗口部件上面的文本，可以通过 `color` 属性指定。

背景色用于绘制窗口部件的填充矩形，可以通过 `background-color` 属性指定。

背景图片使用 `background-image` 属性定义，它用于绘制由 `background-origin` 指定在盒模式中四个区域的图片开始显示的起点位置。背景图片在盒模式域内的对齐和平铺方式可以通过 `background-position` 和 `background-repeat` 属性指定。

如果指定的背景图片具有 `alpha` 通道（透明效果），通过 `background-color` 指定的颜色将会透过透明区域。在 `background-color` 属性中有介绍。

10.（#）

指定某一个按钮，#号后面是指定类的对象名。要知道代码才能运用。

三、语法介绍与问题解决：

1.语法

- 选择器 {
 属性 : 值 ;
}
- QPushButton{

- 选择器：状态{
属性：值
}

- 选择器 :: 辅助控制器 {
 属性 : 值;
}

[illegible]

- 选择器：状态，选择器：状态{
 属性：值;
}

```
QPushButton: hover, QPushButton: pressed {  
    color: red;  
}
```

你可以任意的组合，当然这只是简单的组合，还有更加复杂的组合。

2. 选择器的格式确定后，就要确定你要所要针对的具体的类型，那么就参照 2. 可以应

用样式表的窗口部件。

3. 之后就查看你所要应对的选择器里的部分进行设置，就要再去查看辅助控制器。

4. 再分的细一点，搞清楚一个窗口部件分为几个状态，鼠标划入，点击，关闭...

他们决定了你所要设置状态的属性设置。

5. 定位了前面的，就要改变他的特性了，也就是进行属性设置，查找属性表，对应属性表超找对应的值表，也有可能还要通过值表的迭代（值的值还需要一个方式表达。）最终知道这个值是数字，还是一个颜色，或者是一个 bool 类型，抑或是其他的关键词(如 padding、content...)。仔细阅读他们之间的关系。

2.遇到问题

也许你已经组合了很多的样式表，但是有时候你会发现，有时候有些属性值不起作用，或者说图像变形，并不是你所要看到的效果，不要着急。

1. 查看一下你的语法是否正确，如果你保证确保无误的话，那么就想一想，是否是在构建这个窗口控件之前进行设置的，这个会影响到你的属性是否被读入。参考六、附属例题解释：

32. 定制 QTool 查看是否被其他的属性覆盖。当一个属性被具有同一选择器的几个规则设置时，那么只有最后一个规则起作用（这是一个难点）。

2. 查看是否有相应的关联的属性已被设为 bool=1。

3. 图片无法显示: 查看路径是否正确在 Qt 中是” /” 代替 window 下的” \”，使用相对路径，本应用程序的相对路径（是相应程序读取这个样式 css 文件），当然你也可以写绝对路径的方式读取图片，但是路径这个方法是不提倡的。

4. 大小变化的窗口控件是否背景图片选择了 border-image。详细参考四、高级应用与六、附属例题解释：34. QPushButton 与 image

四、高级应用：

1.九宫格分割法：

之所以叫做九宫格分割法是源于，把边界图分为 3X3 的小格，当填充窗口部件时如图 A，4 个角保持不变，如图 B，

A	B	C
D	E	F
G	H	T

A: 原始图

A	B	C
D	E	F
C	H	T

B: 结果图

其他的 5 个格子被拉伸或者平铺填充可用的空间。

使用 `border-image` 属性可以指定各个边界图，他要求指定一个图像文件名和定义 9 个格子的 4 条“切线”。切线用到上、右、下和左边缘的距离定义。`Border.png` 作为边界图，距离上、右、下和左的切线为 4、8、12 和 16 应该如下定义：

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

但使用边界图时，必须显示地设置 `border-image` 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者缩短。对 `border.png` 应该这样设置指定他的四个边框的宽度：

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

这样才能把刚才切割的四个角的图片完全的放置到 `border` 中当做 `border` 区域的背景。

2.渐变器

Qt 支持三种渐变器查看例题 5.渐变器

- 1.线性渐变 (`qlineargradient`): 连接这两点的线上有一系列的颜色断点。两个控制点之间的不同位置指定不同的颜色。位置用 0 和 1 之间的浮点数来确定，0 对应着第一个控制点，1 对应着第二个控制点。两个指定断点之间的颜色由线性插值得出。
- 2.辐射渐变 (`qradialgradient`): 有一个中心点 (`xn, yn`)，半径 `r`，一个焦点 (`xf, yf`) 以及颜色断点定义。
中心点和半径定义一个圆。颜色从焦点向外扩散，焦点可以是中心点或者圆内的其它点。
- 3.梯形渐变 (`qconicalgradient`): 由一个中心点 (`xn, yn`) 和一个角度 `a` 定义。颜色在中心点周围像钟表的秒针扫过一样扩散。

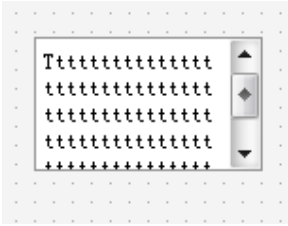
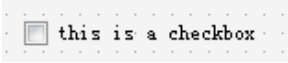


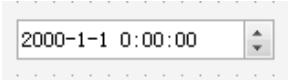

五、样式表参照表：

1.样式选择器类型表

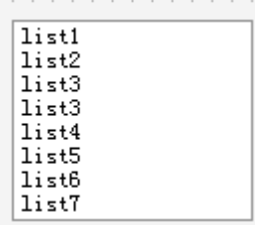
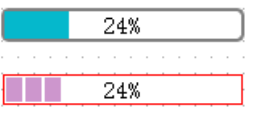

选择器	实例	可以匹配的窗口部件
全局对象	*	对全局的任意窗口部件
类型	QDial	给定类的实例，包括这个类的子类
类	.QDial	给定类的实例，不包括子类
标识	QDial#ageDial	给定对象名称的窗口(ageDial 表示对象的名字，在代码中可知)
Qt 属性	QDial[y="0"]	为某些属性赋值的窗口，表示当 Y (属性) 为 0 (false) 或者为 1 (true) 的时候
子对象	QFrame > QDial	给定窗口部件的直接子类
子孙对象	QFrame QDial	给定窗口部件的子孙窗口部件


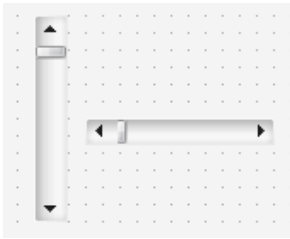
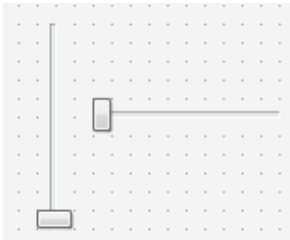

2.可以应用样式表的窗口部件表 = 选择器

部件	可应用的样式
----	--------

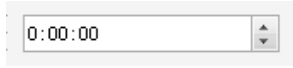

QAbstractScrollArea  <p>上图为一个 QTextEdit 文字不能完全显示而自动添加的滚动条</p>	<ol style="list-style-type: none"> 1. 支持盒模型。 2. 所有 QAbstractScrollArea (QAbstractItemView 、 QGraphicsView、QMdiArea、QPlainTextEdit、QScrollArea、QTextEdit) 的派生类, 包括 QTextEdit 和 QAbstractItemView(带 item 的类), 用 background-attachment 属性支持背景可滚动的, 设置 background-attachment 属性为 fixed 则提供一个 background-image 背景图片不跟随滚动条滚动, 反则设置 background-attachment 属性为 scroll 则是背景图片可以跟随滚动条滚动。 3. 具体看例: 6 定制 QAbstractScrollArea。
QCheckBox 	<ol style="list-style-type: none"> 1. 支持盒模型, 2. 可以用 ::indicator 辅助控制器设置 check 指示符(一个可以选择的小方块), 默认时, check 指示符放置在部件矩形内容的左上角处。 3. spacing 属性指定 check 指示符和文本(右侧的描述文字)的距离。 4. 具体看例: 7 定制 QCheckBox。
QColumnView	<ol style="list-style-type: none"> 1. 可以被 image 属性控制样式, arrow(箭头)指示符可以被 ::left-arrow 和 ::right-arrow 辅助控制器样式控制。
QComboBox 	<ol style="list-style-type: none"> 1. 支持盒模型。 2. 下拉按钮(drop-down button)样式可以被 ::drop-down 辅助控制器设置, 默认情况下, 下拉按钮填充矩形部件的右上侧, 下拉按钮的箭头标记(arrow mark)通过样式 ::down-arrow 辅助控制器控制。默认的情况下, 箭头是位于下拉按钮的中间位置。 3. 具体看例: 8 定制 QComboBox。
QDateEdit 	<p>具体看例: 24 定制 QSpinBox。</p>
QDateTimeEdit 	<p>具体看例: 24 定制 QSpinBox。</p>
QDialog 	<ol style="list-style-type: none"> 1. 只支持 background、background-clip 和 background-origin 属性。 <p>Warning: 确保我们定义了宏 Q_OBJECT 在我们定制的部件里。</p>
QDialogButtonBox <p>上图 QDialog 对话框中的按钮</p>	<p>可以用 button-layout 属性改变按钮的布局。</p>
QDockWidget	<ol style="list-style-type: none"> 1. 当浮动窗口停靠的时候, 支持标题栏和标题栏按钮。 2. border 属性可以控制浮动窗口的边框。

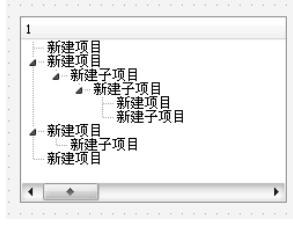
	<p>3. <code>::title</code> 辅助控制器可以定制标题栏样式。</p> <p>4. <code>::title</code> 辅助控制器上的 <code>::close-button</code> 控制浮动窗口关闭按钮属性, <code>float-button</code> 浮动控制按钮。</p> <p>5. 当标题栏可视的时候, <code>:vertical</code> 状态被设置, 当然这个状态取决于浮动窗口的 3 种状态设置。即 <code>:floatable</code>、<code>:closable</code> 和 <code>:movable</code> 状态。</p> <p>6. Note: 用 <code>QMinWindow::separator</code> 去设计大小处理。</p> <p>7. Warning: 设置的样式不能应用于未停靠的窗口上。</p> <p>8. 具体看例: 9 定制 <code>QDockWidget</code>。</p>
QDoubleSpinBox	<p>具体看例: 24 定制 <code>QSpinBox</code>。</p>
QFrame	<p>1. 支持盒模式。</p> <p>2. 自 Qt 版本 4.3 开始, 在 <code>QLabel</code> 上设置一个样式表将自动设置 <code>QFrame::frameStyle</code> 属性到 <code>QFrame::StyledPanel</code>。</p> <p>3. 具体看例: 10 定制 <code>QFrame</code>。</p>
	<p>1. 支持盒模式。</p> <p>2. <code>::title</code> 辅助控制器设置标题样式 (图片中 “GroupBox” 字样)。默认的情况下, 标题位置取决于 <code>QGroupBox::textAlignment</code>。</p> <p>3. 在可点击选择的情况下, 标题包含 check 指示符用 <code>::indicator</code> 辅助控制器设置。</p> <p>4. Spacing 属性是控制 check 指示符与文本描述的空间。</p> <p>5. 具体看例: 11 定制 <code>QGroupBox</code>。</p>
	<p>1. 支持盒模式。</p> <p>2. <code>::section</code> 辅助控制器设置标题视图, section 辅助控制器支持 <code>:middle</code> 、 <code>:first</code> 、 <code>:last</code> 、 <code>:only-one</code> 、 <code>next-selected</code>、<code>:previous-selected</code>、<code>:selected</code> 和 <code>:checked</code> 状态。</p> <p>3. <code>::up-arrow</code> 和 <code>::down-arrow</code> 设置排序的样式。</p> <p>4. 具体看例: 15 定制 <code>HeaderView</code>。</p>
	<p>1. 支持盒模式。</p> <p>2. 不支持鼠标悬停状态(鼠标经过的时候)。</p> <p>3. 自 Qt 版本 4.3 开始, 在 <code>QLabel</code> 上设置一个样式表将自动设置 <code>QFrame::frameStyle</code> 属性到 <code>QFrame::StyledPanel</code>。</p> <p>4. 具体看例: 10 定制 <code>QFrame</code> (<code>QLabel</code> 类继承自 <code>QFrame</code>)。</p>
	<p>1. 支持盒模式。</p> <p>2. <code>selection-color</code> 和 <code>selection-background-color</code> 分别设置选择时的背景颜色和前景色。</p> <p>3. <code>lineEdit-password-character</code> 的属性设置密码类型。</p> <p>4. 具体看例: 13 定制 <code>QLineEdit</code>。</p>
QListView	<p>1. 支持盒模式。</p> <p>2. 当表格交替颜色是可以选择的, 表格交替颜色设置用 <code>alternate-background-color</code> 属性设置。</p> <p>3. 选择的颜色 <code>selection-color</code> 和 <code>selection-background-color</code> 控制选择颜色和选择的背景颜色。</p>

	<p>4. show-decoration-selected 属性控制选择行为。</p> <p>5. ::item 更加细致的控制 <code>QListView</code> 的 <code>items</code>。</p> <p>6. 具体看例： 6 定制 <code>QAbstractScrollArea</code>。</p> <p>具体看例： 14 定制 <code>QListView</code>。</p>
QListWidget 	<p>具体看例： 14 定制 <code>QListView</code>。</p>
QMainWindow (主窗口)	<p>1. 支持样式分割器。</p> <p>2. 在 <code>QMainWindow</code> 用分割器时, <code>QDockWidget</code> 用 ::separator 辅助控制器来设置样式。</p> <p>2. 具体看例： 15 定制 <code>QMainWindow</code>。</p>
QMenu (菜单)	<p>1. 支持盒模式。</p> <p>2. ::item 辅助控制器设置每一个 <code>items</code>, 除此之外还支持状态如 :selected、:default、:exclusive 和 non-exclusive 状态。</p> <p>3. ::indicator 辅助控制器设置菜单的可点击的属性。</p> <p>4. 菜单分隔风格样式由 ::separator 辅助控制器设置。</p> <p>5. right-arrow 和 left-arrow 设置子目录箭头是向右还是向左。</p> <p>6. ::scroller 设置滚动条。</p> <p>7. ::tearoff 设置 <code>tear-off</code> (拆卸下来成为独立的窗口)</p> <p>8. 具体看例： 16 定制 <code>QMenu</code>。</p>
QMenuBar (菜单栏)	<p>1. 支持盒模式。</p> <p>2. spacing 属性设置两个 <code>item</code> 的距离。</p> <p>3. ::item 辅助控制器控制每一个 <code>items</code> 的样式。</p> <p>4. 具体看例： 17 定制 <code>QMenuBar</code>。</p>
QMessageBox (弹出来的报告错处, 提示的对话框)	<p>Messagebox-text-interaction-flags 属性用来修改消息框中的文本</p>
QProgressBar 	<p>1. 支持盒模式。</p> <p>2. ::chunk 辅助控制器设置块状结构样式(进度条里面的颜色设置可能是一段一段的)的进度。条块是显示在盒模时的 <code>content</code> 中。</p> <p>3. 进度条中如果显示文字, text-align 属性设置进度条中文字位置。</p> <p>4. ::indeterminate 不确定状态设置。</p> <p>具体看例： 18 定制 <code>QProgressBar</code>。</p>
QPushButton 	<p>1. 支持盒模式。</p> <p>2. :default、:flat(外框是否突起)、:checked 状态。</p> <p>3. 当按钮在菜单栏里时, 用 ::menu-indicator 辅助控制器设置菜单栏的风格。</p> <p>4. 用 :open 和 :closed 定制是否可点击的按钮。</p> <p>5. 例如: <code>QPushButton{</code></p>

	<pre>background-color: red; border: none;</pre> <pre>}</pre> <p>6. 具体看例： 19 定制 QPushButton。</p>
QRadioButton 	<p>1. 支持盒模式。</p> <p>2. 可以用 <code>::indicator</code> 辅助控制器设置 check 指示符(一个可以选择的小圆圈)，默认时，check 指示符放置在矩形内容的左边。</p> <p>3. <code>spacing</code> 属性指定 check 指示符和文本(右侧的描述文字)的距离。</p> <p>4. 具体看例： 20 定制 QRadioButton。</p>
QScrollBar 	<p>1. 支持盒模式。</p> <p>2. 控件上盒模式横纵的 content 的内容依赖于槽上的滑动器(handle)的移动。</p> <p>3. QScrollBar 的范围(长或者宽由 scroll 的方向确定)由 <code>width</code> 和 <code>height</code> 设置, 用 <code>:horizontal</code> 和 <code>:vertical</code> 设置方向。</p> <p>4. <code>::handle</code>(句柄)辅助控制器宽或者高(设置 <code>min-width</code> 或者 <code>min-height</code>)的最小宽度和最小高度, 取决于它的方向。</p> <p>5. <code>::add-line</code> 设置添加内容的按钮样式。依赖于箭头的方向。<code>::right-arrow</code> 和 <code>::down-arrow</code>, 箭头默认是在 add-line 的盒模式 contents 的中间位置。</p> <p>6. <code>::sub-line</code> 设置减少内容的按钮样式。<code>::left-arrow</code> 和 <code>::up-arrow</code>, 箭头在按钮的中间位置。</p> <p>7. <code>::sub-page</code> 减少页和 <code>add-page</code> 增加页。</p> <p>. 具体看例： 21 定制 QScrollBar。</p>
QSizeGrip (通过它可以改变它所在顶级窗口的大小, 在顶级窗口的右下角)	<p>1. 支持 <code>width</code>、<code>height</code>、<code>image</code> 属性。</p> <p>2. 具体看例： 22 定制 QSizeGrip。</p>
QSlider 	<p>1. 支持盒模式。</p> <p>2. 对于横向, <code>min-width</code> 和 <code>height</code> 属性必须提供。</p> <p>3. 对于纵向, <code>min-height</code> 和 <code>width</code> 属性必须提供。。</p> <p>4. <code>::groove</code> 设置槽, 一般情况下槽是默认在盒模式的 content 中, 控件可以滑动的 handle(句柄)用 <code>::handle</code> 辅助控制器设置。句柄在槽上移动, 在盒模式中的 content。</p> <p>5. 具体看例： 23 定制 QSlider。</p>
QSpinBox 	<p>1. 支持盒模式。</p> <p>2. <code>::up-button</code> 和 <code>up-arrow</code> 控制上面的按钮和按钮上箭头。</p> <p>3. <code>::down-button</code> 和 <code>down-arrow</code> 控制下面的按钮和按钮上的箭头。默认箭头在按钮的中间位置。</p> <p>具体看例： 24 定制 QSpinBox。</p>
QSplitter (界面分离器) 一个界面被分为两个	<p>1. 支持盒模式。</p> <p>2. <code>::handle</code> 设置分割器可移动句柄 handle。</p> <p>具体看例： 25 定制 QSplitter。</p>
QStatusBar (状态栏)	<p>1. 只有 <code>background</code> 属性起作用, 每一个 items 用 <code>::items</code> 设置。</p> <p>2. 具体看例： 26 定制 QStatusBar。</p>

<p>QTabBar</p> 	<ol style="list-style-type: none"> 1. 每一个 tab 页用 <code>::tab</code> 设置样式, 关闭用 <code>::close-button</code>, tab 页支持 <code>only-one</code>、<code>:first</code>、<code>:last</code>、<code>:middle</code>、<code>:previous-selected</code>、<code>:next-selected</code>、<code>:selected</code> 状态。 2. <code>:top</code>、<code>:left</code>、<code>:right</code>、<code>:bottom</code> 状态取决于 tabs 在部件的方向。 3. 选择重叠 tabs 状态 (这个状态时表示在很多 tab 的时候, 点击一个 tab 页为当前显示时, 就表示这个 tab 突出覆盖了左右 tab 的 n 个 px) 是利用负 margins (盒模式中的 margin) 或者是相对位置来控制。 4. 当 tab 很多个的时候, 有时候 tabbar 不能把它完全显示出来, 就会在 tabbar 上显示可以左右调节 tab 页的按钮, 这个按钮为 <code>QToolButton</code>, 定制这个 <code>QToolButton</code> 的宽度用 <code>::scroller</code> 辅助控制器定制。 5. <code>::tear</code> 定制 tear 指示器的样式。 6. <code>QTabBar</code> 的 tab 方向用 <code>alignment</code> (<code>top bottom left right center</code>) 设置。 6. 改变 <code>QTabWidget</code> 的 <code>QTabBar</code> 方向用 <code>tab-bar</code> 辅助控制器。 7. 具体看例: 27 定制 <code>QTabWidget</code> 和 <code>QTabBar</code>。
<p>QTabWidget</p> 	<ol style="list-style-type: none"> 1. 用 <code>::pane</code> 设置页面的外框。 2. <code>::left-corner</code> 和 <code>::right-corner</code> 设置左侧角落和右侧角落。 3. 它的 tab 位置通常用 <code>::tab-bar</code> 设置, 默认是 <code>QWindowsStyle</code> 的风格。 4. <code>:top</code>, <code>:left</code>, <code>:right</code>, <code>:bottom</code> 选择 tab 上、左、下和右的方位。 5. 具体看例: 27 定制 <code>QTabWidget</code> 和 <code>QTabBar</code>。
<p>QTableView</p>	<ol style="list-style-type: none"> 1. 支持盒模式。 2. <code>alternating row colors</code> 可用的时候我们可以使用属性 <code>alternate-background-color</code> 控制交替的表格背景颜色。 3. <code>selection-color</code> 和 <code>selection-background-color</code> 属性设置选择 item 的前景颜色和背景颜色。 4. 在 <code>QTableView</code> 中控件可以作为一个 <code>QAbstractButton</code> 和可以用 <code>QTableView</code> <code>QTableCornerButton::section</code> 设置样式。 5. 表格颜色 <code>gridline-color</code>。 6. 参看 <code>QAbsractScrollArea</code> 设置背景 <code>backgrounds</code>。 <p>具体看例: 28 定制 <code>QTableView</code>。</p>
<p>QTableWidget</p> 	<p>具体看例: 28 定制 <code>QTableView</code>。</p>
<p>QTextEdit (一个文本编辑器)</p>	<ol style="list-style-type: none"> 1. 支持盒模式。 2. <code>selection-color</code> 和 <code>selection-background-color</code> 决定它的文本颜色和背景颜色。

	具体看例： 6 定制 QAbstractScrollArea。
QTimeEdit 	具体看例： 24 定制 QSpinBox。
QToolBar (工具栏)	1. 支持盒模式。 2. :top, :left, :right, :bottom 工具栏的位置, 是在主窗口的上, 左, 右, 下状态设置属性。 3. :first, :last, :middle, :only-one 选择了工具栏中第一组按钮。 4. 工具栏的分隔符是由 ::separator 控制。 5. 可以控制工具栏可移动: ::handle 选择器选择它并改变样式。 6. 具体看例： 29 定制 QToolBar。
QToolButton (工具栏按钮)	1. 支持盒模式。 2. 如果 QToolBox 有一个菜单, ::menu-indicator 来选择它的指示符 (就是可以移动工具栏的句柄, 可以参考 WPS 的新建工具栏)。 3. 当 QToolBar 在 QToolButton::MenuButtonPopup 模式下, ::menu-button 控制这个菜单按钮, ::menu-arrow 是控制这个按钮的箭头, 默认的情况下会放置在按钮上 content 位置。 4. 当 QToolBar 显示箭头时候: ::up-arrow, ::down-arrow, ::left-arrow and ::right-arrow 可以被用到。 5. 具体看例： 30 定制 QToolButton。
QToolBox  如同好友 QQ 模式	1. 支持盒模式。 2. ::tab 辅助控制器设置 tab, tab 支持: only-one、:first、:last、:middle、previous-selected:next-selected、:selected 状态。
QToolTip (鼠标划过时的提示语)	1. 支持盒模式。 2. 支持 opacity (不透明) 属性。 具体看例： 10 定制 QFrame (QToolTip 类继承自 QFrame) 31 定制 QToolTip。
QTreeView	1. 支持盒模式。 2. 当 alternating row colors (行交替着不同的颜色) 是可用的时候, 可以通过 alternate-background-color 属性设置它交替颜色。 3. selection-color 属性和 selection-background-color 属性设置选择 item 的前景颜色和背景颜色。 4. show-decoration-selected 设置选择行为属性。 5. 分支辅助控制器: ::branch 支持: open, :closed, :has-sibling and :has-children 状态。

	<p>6. <code>::item</code> 辅助控制器更详细的控制 <code>QTreeView</code> 的 <code>items</code>。</p> <p>7. 看 <code>QAbstractScrollArea</code> 关于可滚动背景的描述。</p> <p>具体看例： 32 定制 <code>QTreeView</code>。</p>
QTreeWidget 	<p>具体看例： 32 定制 <code>QTreeView</code>。</p>
QWidget	<p>1. 只支持 <code>background</code>, <code>background-clip</code> 和 <code>background-origin</code> 属性。</p> <p>2. 如果你的类是 <code>QWidget</code> 的子类, 你需要重写 <code>paintEvent</code> 函数如下:</p> <pre>void CustomWidget::paintEvent(QPaintEvent *) { QStyleOption opt; opt.init(this); QPainter p(this); style()->drawPrimitive(QStyle::PE_Widget, &opt, &p, this); }</pre> <p>/*如果没有样式表设置, 那么它是什么作用都不起的*/</p> <p>Warning: 确保我们定义了宏 <code>Q_OBJECT</code> 在我们定制的部件里。</p>

3. 辅助控制器列表

辅助控制器	说明
<code>::add-line</code>	在 <code>QScrollBar</code> 中添加一行的按钮。
<code>::add-page</code>	在 <code>QScrollBar</code> 中添加页的按钮。
<code>::branch</code>	<code>QTreeView</code> 的分支。
<code>::chunk</code>	<code>QProgressBar</code> 的进度条里的块 (里面的进度条可以变成一块一块的增加, 而不是整体都是一种颜色)。
<code>::colse-button</code>	<code>QDockWidget</code> 和 <code>QTabBar</code> 标题栏上的关闭按钮。
<code>::corner</code>	在 <code>QAbstractScrollArea</code> 两个滚动条之间的位置。
<code>::drwn-arrow</code>	<code>QComboBox</code> 、 <code>QHeaderView</code> (排序时需要)、 <code>QScrollBar</code> 、 <code>QSpinBox</code> 的向下箭头。
<code>::down-button</code>	<code>QScrollBar</code> 、 <code>QSpinBox</code> 的向下按钮。
<code>::drop-down</code>	<code>QComboBox</code> 展开时。
<code>::float-button</code>	<code>QDockWidget</code> 标题栏上的浮动按钮。
<code>::groove</code>	<code>QSlider</code> 的槽。
<code>::indicator</code>	<code>QAbstractItemView</code> 、 <code>QCheckBox</code> 、 <code>QRadioButton</code> 、可点击的 <code>QMenu</code> 的 <code>item</code> 、可点击的 <code>QGroupBox</code> 的指示符。

::handle	QScrollBar、QSplitter、QSlider 的滑块。
::icon	QAbstractitemView 和 QMenu 的图标。
::item	QAbstractitemView、QMenuBar、QMenu、QStatusBar 的单独的一项。
::left-arrow	QScrollBar 的向左的箭头。
::left-corner	QTabWidget 的左侧。
::menu-arrow	菜单里 QToolButton 箭头。
::menu-button	工具栏上的按钮。
::menu-indicator	菜单里的 QPushButton 指示符。
::right-arrow	QMenu 或者是 QScrollBar 的右侧箭头。
::pane	QTabWidget 去掉标题的框架。
::right-corner	QTabWidget 的右侧。
::scroller	QMenu 和 QTabBar 因为界面大小布局左右调试的滚动按钮。
::section	QHeaderView 的表头横向和纵向。
::separator	QMainWindow 和 QMenu 的分离器（就是一个主窗口被分割成几个小的区域的线，QMenu 里是 item 的分离线）。
::sub-line	QScrollBar 内容减少方向的按钮。
::sub-page	QScrollBar 减少一页的按钮，在滑块与减少一行 sub-line 之间。
::tab	QTabBar 和 QToolBox 的一个页选项。
::tab-bar	一个 QTabWidget 的 tab 按钮，设置 tabs 一般用::tab。
::tear	TabBar 的指示符
::tearoff	QMenu 的指示符
::text	QAbstractitemView 的内容。
::title	QGroupBox 和 QDockWidget 的标题。
::up-arrow	QHeaderView（排序时）、QScrollBar、QSpinBox 向上按钮箭头。
::up-button	QSpinBox 的向上按钮。

4. 状态列表

状态	说明
:active	当前活动的窗口。
:adjoins-item	QTreeView 的一个 branch 存在毗邻下一个与自己不是兄弟项目的项。
:alternate	当 QAbstractitemView 的 QAbstractitemView : : alternatingRowColors() 的属性设置为 true 时，行之间背景颜色交替颜色变化，
:bottom	在 item 的下面，例如 QTabBar 的 tab 按钮在下面。
:checked	按钮已选中。
:closable	items 是可以关闭的，例如 QDockWidget 有一个 QDockWidget:: DockWidgetClosable 的功能。
:closed	（open 相对的）窗口位于关闭或者销毁的状态，例如 QTreeView 没有打开时的状态

:default	默认的状态。
:disabled	禁用窗口部件
:editable	QComboBox 可以编辑的
:edit-focus	那种可编辑的控件，比如文本框，当它正在编辑的时候，就是 QStyle::State_HasEditFocus 状态。
:enabled	启用窗口部件。
:exclusive	表示按钮组设置为单选，只能选择一个的状态。例如菜单栏的选项。
:first	部件的第一个，例如 QTabBar 的第一个 tab
:flat	没有突起的部件。
:floatable	部件是可浮动，例如 QDockWidget。
:focus	窗口部件有输入焦点。
:has-children	Item 有子目录的，例如 QTreeView。
:has-siblings	有兄弟目录的，例如 QTreeView。
:horizontal	部件是横向的。
:hover	鼠标划过窗口部件时的状态。
:indeterminate	按钮部分被选中的状态。
:last	最后一个，例如：QTabBar 的最后一个 tab。
:left	位于左面，例如 QTabBar 的 tab 位于左面的那个。
:maximized	最大化状态。
:middle	在列表中中间位置，例如 QTabBar 的 tab 不是最后一个也不是第一个的。
:minimized	最小化的时候。
:movable	这个部件可以移动，例如 QDockWidget。
:no-frame	这个部件是没有 frame 的，例如 QLineEdit 和 QSpinBox。
:non-exclusive	不能单选的，菜单来就是单选的。
:off	前提是这个部件是可以选或者不选的，与 on 相对，窗口部件是“off”状态。
:on	前提是这个部件是可以选或者不选的，与 off 相对，窗口部件是“on”状态。
:only-on	只有一个，例如 QTabBar 只有一个 tab 的时候。
:open	窗口部件位于打开或者是展开的状态，如 QTreeView, QComboBox 和 QPushButton 在 menu 时的打开状态。
:next-selected	选择下一个的时候，在列表中，例如 QTabBar。
:pressed	鼠标按键点击窗口部件
:previous-selected	选择上一个，例如 QTabBar。
:read-only	部件是只读的，例如 QLineEdit, 和 QComboBox 的编辑窗口为不可编辑时。
:right	部件位置的右侧，QTabBar 的 tab 可以位于右侧。
:selected	被选择的，例如 QTabWidget 被选择的 tab 页或者是 QMenu 下拉菜单被选择的某一个条目。
:top	位于部件的上面，例如 QTabBar 的 tab 可以位于上面，也可以在下面一样。
:unchecked	按钮未被选中的。

:vertical	垂直的方向。
:window	这个部件是一个窗口。

5. 属性列表

属性	值	说明
alternate-background-color	Brush	表示交替论表格颜色的属性，它应用于 QAbstractItemView 的子类，如果你不设置这个颜色值，它会有一个默认的值备用。 <pre>QTreeView { alternate-background-color: blue; background: yellow; }</pre> 你也可以参考 background 和 selection-background-color
background	Background	简单的设置背景颜色，等价于 background-color, background-image, background-repeat 或者 background-position. 这个属性支持 QAbstractItemView 的子类，子类包括（QAbstractSpinBox 子类, QCheckBox, QDialog, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, QToolTip, 和 plain 的 QWidget） 通常，它需要设置 Qt::BrushStyle 样式一个填充模式，你可以用 background-color 属性为三个渐变器来和 Qt::SolidPattern 设置样式。 <pre>QLabel { background-color: yellow } QLineEdit { background-color: rgb(255, 0, 0) }</pre> 可以参考： background-origin, selection-background-color, background-clip, background-attachment, alternate-background-color.
background-color	Brush	用于部件的背景颜色。 例如： <pre>QLabel { background-color: yellow } QLineEdit { background-color: rgb(255, 0, 0) }</pre>
background-image	Url	用于部件的背景图片，可以用 background-color 的般透明度给图片加效果。 <pre>QFrame {</pre>

		<pre>background-image: url(:/images/hydro.png) }</pre>
background-repeat	Repeat	<p>无论是否指定重复的填充背景图片 (background-image) 于盒模式的矩形内, 只要图片太小它都会自动的重复填充。如果这个 y 或者 x 属性没有被指定, 背景图片是从两边 (x. y) 同时开始重复。例:</p> <pre>QFrame { background: white url(e:/wb/styles/start.png); background-repeat: repeat-y; background-position: right; }</pre> <p>这样我们指定了背景图片它重复的方向是 y (竖着) 轴, 右侧开始。</p>
background-position	Alignment	<p>在盒模式中定位背景图片开始显示的位置, 默认是左上角开始显示。</p> <p>例如:</p> <pre>QFrame { background: url(:/images/footer.png); background-position: bottom left; }</pre> <p>这样就可以设置部件显示图片是从左下角开始显示。</p>
background-attachment	Attachment	<p>描述背景图片是否在 QAbstractScrollArea 的区域跟随滚动条滚动, scrolled 是图片跟随滚动条滚动, fixed 是图片不跟随滚动条滚动。</p> <pre>QTextEdit { background-image: url("leaves.png"); background-attachment: fixed; }</pre> <p>可以参考 background</p>
background-clip	Origin	<ol style="list-style-type: none"> 1. 在部件的盒模式中那些可以当做背景显示的。 2. 这个属性指定盒模式的 background-color 和 background-image 被裁减。 3. 这个属性支持以下类 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QDialog, QFrame, QGroupBox, QLabel, QPushButton, QRadioButton, QSplitter, QTextEdit, QToolTip, 和一些无复杂设置的 QWidgetets. 4. 如果没有使用该属性, 那么默认情况下是

		<p>border。</p> <p>5. 例如:</p> <pre>QFrame { background-image: url(:/images/header.png); background-position: top left; background-origin: content; background-clip: padding; }</pre> <p>6. 可参考 background, background-origin 和 The Box Model.</p>
background-origin	Origin	<p>1. 可以 background-position 和 background-image 联合一起使用设置空间的背景显示在盒模式中的起点位置。</p> <p>2. 默认是支持类 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QDialog, QFrame, QGroupBox, QLabel, QPushButton, QRadioButton, QSplitter, QTextEdit, QToolTip, 和一些设置比较单一的 QWidget.</p> <p>3. 默认情况下是 padding。</p> <p>4. 例子:</p> <pre>QFrame { background-image: url(:/images/header.png); background-position: top left; background-origin: content; }</pre> <p>5. 可以参考 background 和 The Box Model.</p>
border	Border	<p>1. 简便的设置部件的外框线。</p> <p>2. 等价于 border-color, border-style, 或者 border-width.</p> <p>5. 这个属性支持 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, QToolTip, 和一些设置比较单一的 QWidget.</p> <p>3. 例子:</p> <pre>QLineEdit { border: 1px solid white }</pre>
border-top	Border	简单的设置部件上的外框线, 等效于 border-top-color, border-top-style, 或者 border-top-width
border-right	Border	简单的设置部件上的右外框线, 等效于

		border-right-color, border-right-style, 或者 border-right-width
border-bottom	Border	简单的设置部件上的左外框线，等效于 border-bottom-color, border-bottom-style, 或者 border-bottom-width
border-left	Border	简单的设置部件的下外框线，等效于 border-left-color, border-left-style, 或者 border-left-width
border-color	Box Colors	<p>1. 设置所有外框线的颜色, border-top-color, border-right-color, border-bottom-color, 和 border-left-color.</p> <p>2. 支持所有的 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, QToolTip, 和一些设置比较单一的 QWidget.</p> <p>3. 默认情况下一般默认为和前景色一致。</p> <p>4. 例子:</p> <pre>QLineEdit { border-width: 1px; border-style: solid; border-color: white; }</pre> <p>5. 可以参考 border-style, border-width, border-image, and The Box Model</p>
border-top-color	Brush	外框线的上面颜色。
border-right-color	Brush	外框线的右面颜色。
border-bottom-color	Brush	外框线的下面颜色。
border-left-color	Brush	外框线的左面颜色。
border-image	Border Image	<p>1. 把图片的分割的九个区域中的四个角区域填充到 border 中（详细查看九宫格分割法），其实 border 在盒模式中是一块区域，默认值只是因为太小了，所以我们通常认为是线，但是在这里我们要改正一下我们以往的惯用思维。</p> <p>2. 支持所有 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit and QToolTip.</p> <p>3. 可以参考 border-color, border-style, border-width, an 和 The Box Model.</p>
border-radius	Radius	1. 外框线 border 角的弧度，

		<p>2. 等效于 border-top-left-radius, border-top-right-radius, border-bottom-right-radius, 和 border-bottom-left-radius.</p> <p>3. 默认的情况下是 0px.</p> <p>4. 例子: QLineEdit { border-width: 1px; border-style: solid; border-radius: 4px; }</p> <p>5. 可以查看 border-width 和 The Box Model.</p>
border-top-left-radius	Radius	左上角角的弧度。
border-top-right-radius	Radius	右上角角的弧度。
border-bottom-right-radius	Radius	左下角角的弧度。
border-bottom-left-radius	Radius	右下角角的弧度。
border-style	Border Style	<p>1. 对所有的 border 设置。</p> <p>2. 支持所有的类 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, 和 QToolTip.</p> <p>3. 默认情况下是 none。</p> <p>4. 例子: QLineEdit { border-width: 1px; border-style: solid; border-color: blue; }</p> <p>5. 可以查看 border-color, border-style, border-image, 和 The Box Model.</p>
border-top-style	Border Style	border 的上外框的样式。
border-right-style	Border Style	border 的右外框的样式。
border-bottom-style	Border Style	border 的下外框的样式。
border-left-style	Border Style	border 的左外框的样式。

border-width	Box Lengths	<p>1. border 的宽度值。</p> <p>2. 等效于 border-top-width, border-right-width, border-bottom-width, 和 border-left-width.</p> <p>3. 支持所有的类 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, 和 QToolTip.</p> <p>4. 例子 <pre>QLineEdit { border-width: 2px; border-style: solid; border-color: darkblue; }</pre> </p> <p>5. 可以参考 border-color, border-radius, border-style, border-image, 和 The Box Model.</p>
border-top-width	Length	border 的上边缘线的宽度。
border-right-width	Length	border 的右边缘线的宽度。
border-bottom-width	Length	border 的下边缘线的宽度。
border-left-width	Length	border 的左边缘线的宽度。
bottom	Length	<p>1. 相对位置下（默认的情况下）辅助控制器的移动就抵消了它原来的位置，指定 bottom y 等效于 top: -y。（看 TabWidget）。</p> <p>2. 如果位置是绝对的，辅助控制器的 bottom 关联于母控件的边界。</p> <p>3. 例子： <pre>QSpinBox::down-button { bottom: 2px }</pre> </p> <p>4. 可以参考 left, right, and top.</p>
button-layout	Number	<p>1. 对类 QDialogButtonBox 和 QMessageBox 的按钮进行布局。</p> <p>2. 提供的值是 0 (WinLayout), 1 (MacLayout), 2 (KdeLayout), 3 (GnomeLayout)（不同系统值不一样）</p> <p>3. 默认情况下是当前情况下 SH_DialogButtonLayout 的风格。</p> <p>4. 例子： <pre>* { button-layout: 2 }</pre> </p>
color	Brush	<p>1. 字体的颜色，认为是前景色。</p> <p>2. 支持 QWidget:: palette 都可以支持的属性。</p> <p>3. 这个属性没有被设置默认为黑色。</p>

		<p>4. 例子:</p> <pre>QPushButton { color: red }</pre> <p>5. 可以参看 background 和 selection-color.</p>
dialogbuttonbox-buttons-have-icons	Boolean	<p>1. 是否显示按钮图标。</p> <p>2. 设置为 1 则显示图标, 设置为 0 则图标不显示。</p> <p>3. 例子:</p> <pre>QDialogButtonBox { dialogbuttonbox-buttons-have-icons: 1; }</pre> <p>Note: 在使用这个属性之前一定要确保 QDialogButtonBox 被创建, 所以最好应用在父界面上或者主程序里。</p>
font	Font	<p>1. 简单的对界面的字体进行设置。</p> <p>2. 等效于 font-family, font-size, font-style, 及 font-weight.</p> <p>3. 那么就支持所有的部件 (部件支持 font) 使用。</p> <p>4. 默认的情况下是 QWidget::font.</p> <p>5. 例子:</p> <pre>QCheckBox { font: bold italic large "Times New Roman" }</pre>
font-family	String	<p>1. 字体集</p> <p>2. 例子:</p> <pre>QCheckBox { font-family: "New Century Schoolbook" }</pre>
font-size	Font Size	<p>1. 文本字体的大小。</p> <p>2. 在 Qt 中只支持 pt 和 px.</p> <p>3. 例子:</p> <pre>QTextEdit { font-size: 12px }</pre>
font-style	Font Style	<p>1. 文本字体的样式。</p> <p>2. 例子:</p> <pre>QTextEdit { font-style: italic }</pre>
font-weight	Font Weight	文本字体的磅数。
gridline-color*	Color	<p>1. 在 QTableView 的表格颜色。</p> <p>2. 如果这个属相没被设定, 默认值被当前样式 SH_Table_GridLineColor 指定。</p> <p>3. 例子:</p> <pre>* { gridline-color: gray }</pre>
height	Length	<p>1. 辅助控制器的高度 (例如一个 widget)。</p> <p>2. 如果没有设定, 则默认值是根据当前辅助控制器自己的样式或者是窗口部件的样式显示。</p> <p>3. Warning: 如果一个部件设定了固定的宽和高,</p>

		<p>那么这个属性将不起作用，如设置了一个部件的 min-height 和 max-height。</p> <p>4. 例子： <code>QSpinBox::down-button { height: 10px }</code></p> <p>5. 可以参考 width。</p>
icon-size	Length	<p>1. 在部件中设置图标的高和宽。</p> <p>2. 以下的类可以用到这个属性</p> <p> <code>QCheckBox</code> <code>QListView</code> <code>QPushButton</code> <code>QRadioButton</code> <code>QTabBar</code> <code>QToolBar</code> <code>QToolBox</code> <code>QTreeView</code> </p>
image*	Url+	<p>1. 在辅助控制器的盒模式中的 content 中绘制图形。</p> <p>2. 这个属性接受一个 Urls 列表或者是一个 svg</p> <p>3. 设置了一个辅助控制器的图片，默认的情况下是设置了这个选择器的宽和高。</p> <p>4. 4.3 中和更高的版本中，可以指定图片开始显示位置。</p> <p>5. 对于辅助控制器的 noly-将不支持其他的类型元素。</p> <p>6. Warning: QIcon 的 SVG 插件需要的是 SVG 图片。</p> <p>7. <code>QSpinBox::down-button{</code> <code>image: url(../images/spindown.png)</code> <code>}</code> </p>
image-position	alignment	<p>在 4.3 中和更高的版本中，可以指定图片开始显示位置。</p>
left	Length	<p>1. 相对位置下（默认）辅助控制器的移动就抵消了它原来的位置向右。</p> <p>2. 绝对位置下，关联于母控件的 left。</p> <p>3. 默认的情况下是 0。</p> <p>4. 例子： <code>QSpinBox::down-button { left: 2px }</code></p> <p>5. 可以参考 right, top, 和 bottom.</p>
lineEdit-password-character*	Number	<p>1. QLineEdit 的密码，是 Unicode 格式。</p> <p>2. 默认是 SH_LineEdit_PasswordCharacter 属性的样式。</p> <p>3. 例子： <code>* { lineedit-password-character: 9679 }</code> </p>
margin	Box Lengths	<p>1. 部件的间距。</p>

		<p>2. 等效于 margin-top, margin-right, margin-bottom, and margin-left.</p> <p>3. 支持的类有 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, 和 QToolTip.</p> <p>4. 默认是 0.</p> <p>5. 例子:</p> <pre>QLineEdit { margin: 2px }</pre> <p>6. 可以参考 padding, spacing, The Box Model.</p>
margin-top	Length	上面的空白区域。
margin-right	Length	右面的空白区域。
margin-bottom	Length	下面的空白区域。
margin-left	Length	左面的空白区域。
max-height	Length	<p>1. 辅选择控制器的最大高度。</p> <p>2. 支持所有的 QAbstractItemView 所有子类 QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSizeGrip, QSpinBox, QSplitter, QStatusBar, QTextEdit, 和 QToolTip.</p> <p>3. 默认只依赖于当前的部件盒模型的 content.</p> <p>4. 例子: QSpinBox { max-height: 24px }</p> <p>5. 可以参考 max-width.</p>
max-width	Length	<p>1. 辅选择控制器的最大宽度。</p> <p>2. 支持所有的 QAbstractItemView 所有子类 QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSizeGrip, QSpinBox, QSplitter, QStatusBar, QTextEdit, 和 QToolTip.</p> <p>3. 默认只依赖于当前的部件盒模型的 content.</p> <p>4. 例子: QSpinBox { max-width: 72px }</p> <p>5. 可以参考 max-height.</p>
messagebox-text-interaction-flags*	Number	<p>1. 一个文本交互的一个行为。它的值依赖于 Qt::TextInteractionFlags.</p> <p>2. 默认是 SH_MessageBox_TextInteractionFlags 样式。</p> <p>3. 例子:</p> <pre>QMessageBox {</pre>

		<p>messagebox-text-interaction-flags: 5 }</p>
min-height	Length	<p>1. 辅选择控制器的最小高度。</p> <p>2. 支持所有的 QAbstractItemView 所有子类 QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSizeGrip, QSpinBox, QSplitter, QStatusBar, QTextEdit, 和 QToolTip.</p> <p>3 默认只依赖于当前的部件和模型的 contents。</p> <p>4 例子: QSpinBox { mix-height: 24px }</p> <p>5. 可以参考 mix-width.</p>
min-width	Length	<p>1. 辅选择控制器的最小宽度。</p> <p>2. 支持所有的 QAbstractItemView 所有子类 QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSizeGrip, QSpinBox, QSplitter, QStatusBar, QTextEdit, 和 QToolTip.</p> <p>3. 默认只依赖于当前的部件盒模型的 content。</p> <p>6. 例子: QSpinBox { max-widht: 72px }</p> <p>可以参考 min-height.</p>
opacity*	Number	<p>1. 部件的不透明度, (0[transparent]~255[opaque]).</p> <p>2. 只支持 tooltips</p> <p>3. 如果没有设置样式则依赖于 SH_ToolTipLabel_Opacity。</p> <p>4. 例子: QToolTip { opacity: 223 }</p>
padding	Box Lengths	<p>1. 部件的 padding。</p> <p>2. 等效于 padding-top, padding-right, padding-bottom, 和 padding-left。</p> <p>3. 支持类 QAbstractItemView 的子类, QAbstractSpinBox subclasses, QCheckBox, QComboBox, QFrame, QGroupBox, QLabel, QLineEdit, QMenu, QMenuBar, QPushButton, QRadioButton, QSplitter, QTextEdit, 和 QToolTip.</p> <p>4. 例子: QLineEdit { padding: 3px }</p> <p>5. 可以参考 margin, spacing, 和 The Box Model.</p>

padding-top	Length	部件的上面的 padding。
padding-right	Length	部件的右面的 padding。
padding-bottom	Length	部件的下面的 padding。
padding-left	Length	部件的左面的 padding。
paint-alternating-row-colors-for-empty-area	bool	控制是否可以绘制颜色交替的 QTreeView 的空白区域。
position	relative absolute	用相对的还是绝对的控制 left, right, top 和 bottom 的位置，默认的情况下是相对的。
right	Length	<ol style="list-style-type: none"> 1. 如果是相对位置（默认）辅助控制器的移动抵消了原来的位置向左。 2. 绝对位置，子控件的右边界关联于母控件的右边界。 3. 默认的情况下是 0。 4. 例子： QSpinBox::down-button { right: 2px } 5. 可以参考 left, top, 和 bottom.
selection-background-color*	Brush	<ol style="list-style-type: none"> 1. 选择一个 item 时的背景颜色。 2. 这个部件是可选并且 QWidget::palette 都支持此属性。 3. 默认情况下是 Highlight。 4. 例子： QTextEdit { selection-background-color: darkblue } 5. 可以参考 selection-color 和 background.
selection-color*	Brush	<ol style="list-style-type: none"> 1. 选择一个 item 时的前景色。 2. 这个部件是可选并且 QWidget::palette 都支持此属性。 3. 默认是 HighlightedText 4. 例子： QTextEdit { selection-color: white } 5. 可以查看 selection-background-color and color.
show-decoration-selected*	Boolean	<ol style="list-style-type: none"> 1. 2. 默认情况下 SH_ItemView_ShowDecorationSelected 控制样式。 3. 例子： * { show-decoration-selected: 1 }
spacing*	Length	<ol style="list-style-type: none"> 1. 部件的内部空间大小。 2. 支持类 QCheckBox, checkable QGroupBoxes, QMenuBar, and QRadioButton。 3. 默认值由当前的部件的大小决定。 4. 例子： QMenuBar { spacing: 10 }

		5. 可以参考 padding 和 margin.
subcontrol-origin*	Origin	<p>1. 在盒模式的四个区域中设置辅助控制器起点位置。</p> <p>2. 默认为 padding。</p> <p>3. 例子：</p> <pre>QSpinBox::up-button { image: url(/images/spinup.png); subcontrol-origin: content; subcontrol-position: right top; }</pre> <p>4. 可以参考 subcontrol-position.</p>
text-align	Alignment	<p>1. 设置文本，图标在部件盒模式 content 中文本起始方向。</p> <p>2. 例子：</p> <pre>QPushButton { text-align: left; }</pre> <p>Note: 这个属性目前（4.8）只支持 QPushButton 和 QProgressBar.</p>
text-decoration	none underline overline line-through	添加文本效果。没有效果，下划线，上划线，线穿过文本。
top	Length	<p>1. 如果是相对位置（默认）辅助控制器的移动通过抵消原来的位置向下移动。</p> <p>2. 绝对位置，子控制器的上边界关联于母控件的上边界线。</p> <p>3. 默认的情况下是 0.</p> <p>4. 例子：</p> <pre>QSpinBox::up-button { top: 2px }</pre> <p>5. 可以参考 right, left, 和 bottom.</p>
width	Length	<p>1. 辅助控制器的宽度（例如一个 widget）。</p> <p>2. 如果没有设定，则默认值是根据当前辅助控制器自己的样式显示。</p> <p>3. Warning: 如果一个部件设定了固定的宽和高，那么这个属性会不起作用的，如设置了一个部件的 min-height 和 max-height。</p> <p>4. 例子：</p> <pre>QSpinBox::up-button { width: 12px }</pre> <p>5. 可以参考 height.</p>

6. 图标列表

暂不介绍。

5. 值列表

阅读此表方式：

这是属性的值，通过属性查找到这里时就按照值查找后面对应的语法，把语法里的黑体的首字母非大写的直接加在后面。语法一栏里仍然有黑体字体，仍然按照这个方式查找，直到没有黑色首字母大写值，参看例题。

值	语法	说明
Alignment	{ top bottom left right center }	方位
Attachment	{ scroll fixed }*	背景是否跟随滚动条滚动
Background	{ Brush Url Repeat Alignment }*	任意的一个代替在查找下一层（笔刷，路径，重复，方位）。
Border	{ Border Style Length Brush }*	简单的设置线的属性，长度，笔刷，
Boolean	0 1	True (1) false (0)
Border Image	none Url Number {4} (stretch repeat) {0,2}	看九宫格分割法
Border Style	dashed dot-dash dot-dot-dash dotted double groove inset outset ridge solid none	用这些值绘制一个 border
Box Colors	Brush {1,4}	可以设置四个方向的颜色顺序为上，右，下，左，缺

		省方式参考下面 Box Lengths 。
Box Lengths	Length {1, 4}	<p>四个方向顺序为 top, right, bottom, and left 注释皆为此顺序。</p> <p>例：</p> <p>1. QLabel { border-width: 1px } /* 1px 1px 1px 1px */</p> <p>当输入一个值如 1px 则默认 QLabel 的四个方向的值都为 1px。</p> <p>此时是 right 默认用 top 的值，bottom 自动用 top 的值，right 自动用 right 的</p> <p>2. QLabel { border-width: 1px 2px } /* 1px 2px 1px 2px */</p> <p>输入两个值则表示 bottom 值和 right 的都使用默认的值 bottom 使用 top，right 使用 left。</p> <p>3. QLabel { border-width: 1px 2px 3px } /* 1px 2px 3px 2px */</p> <p>输入三个值表示最后一个 left 值默认用 right 侧的值</p> <p>4. QLabel { border-width: 1px 2px 3px 4px } /* 1px 2px 3px 4px */</p>
Brush	Color Gradient PaletteRole	指定颜色 渐变 调色板
Color	rgb(r, g, b) rgba(r, g, b, a) hsv(h, s, v) hsla(h, s, v, a) #rrggbb Color Name	颜色 rgba 和 hsla 后面的参数是透明度。
Font	(Font Style Font Weight) {0, 2} Font Size String	简单设置文字字体
Font Size	Length	文字字体大小
Font Style	normal italic oblique	文字字体的格式
FontWeight	normal bold 100	文字的磅 bold 加粗

	200 ... 900	
Gradient	qlineargradient qradialgradient qconicalgradient	渐变器（ 线性的渐变 径向渐变（辐射渐变） 梯形渐变） 边界的模式是左上角（0,0），右下角（1,1）参数是从 0 到 1，一般为实际的盒模式的坐标。必须是升序排序。 例： <pre>/* linear gradient from white to green */ QTextEdit { background:qlineargradient(x1:0, y1:0, x2:1, y2:1,stop:0 white, stop: 0.4 gray, stop:1 green) } ((x1, y2) (x2, y2) 线性的两个点) /* linear gradient from white to green */ QTextEdit { background:qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 white, stop: 0.4 rgba(10, 20, 30, 40),stop:1 rgb(0, 200, 230, 200)) (透明度) } /* conical gradient from white to green */ QTextEdit { background:qconicalgradient(cx:0.5,cy:0.5,angle:30,stop:0 white, stop:1 #00FF00) } /*中心点 (0.5,0.5) 角度 30*/ /* radial gradient from white to green */ QTextEdit { background:qradialgradient(cx:0, cy:0, radius:1,fx:0.5, fy:0.5, stop:0 white, stop:1 green) } /*中心点 (0, 0)，半径 1 焦点 (0.5,0.5) 从焦点开始渐变*/</pre>
Icon	(Url (disabled active normal selected) ?	一组图标地址 + 按钮的（禁用 活动 正常 选择） ? （on off） 意思是前面括号里四个中的一个状态和后面两个中的一个的组合来控制图片的显示哪一个。 例： <pre>* {</pre>

	(on off)?)*	<pre> file-icon: url(file.png), url(file_selected.png) selected; } MessageBox { dialogbuttonbox-buttons-have-icons: true; dialog-ok-icon: url(ok.svg); dialog-cancel-icon: url(cancel.png), url(grayed_cancel.png) disabled; } </pre>
Length	Number (px pt em ex)?	长度单位（数字+其中一个）
Number	A decimal integer or a real number	一些十进制的整数或者是真假值，数字如：0 ， 18， +127， -255， 12.34， -.5， 0008 等。
Origin	margin border padding content	margin: 最外边的矩形。 可以控制两个部件之间的 空隙。 border: 为边界预留的空间。 padding: 在 border 里面控制 border 与部件内容的 空间。 content: 部件窗口内容。 可以参考 The Box Model.
PaletteRole	alternate-base base bright-text button button-text dark highlight highlighted-text light link link-visited mid midlight shadow text window window-text	例子: QPushButton { color: palette(dark); }
Radius	Length {1, 2}	角的弧度 1. 可以是一个值这样就是角的直接弧度。 2. 两个值的时候，第一个是横向的弧度，第二个是纵

		向的弧度。可以制作出一个类似椭圆四分之一圆的角度。
Repeat	repeat-x repeat-y repeat no-repeat	repeat-x: 横向重复图. repeat-y: 纵向重复. repeat: 横纵同时. no-repeat: 不重复.
Url	url(filename)	地址

六、附属例题解释：惯用方法

1. 定制全局颜色值

```
*{
background:blue;
color:red;
}
```

2. 定制按钮(QPushButton)

```
1. QPushButton{
    background-color:red;
}
/*imagesForExample: example_for_QPushButton.png*/
```



```
2. QPushButton{
    background-color:red;
    border-style: outset;
    border-width:2px;
    border-color:beige;
}/*imagesForExample: example_for_QPushButton2.png*/
```



```

3. QPushButton {
    background-color: red;
    border-style: outset;
    border-width: 2px;
    border-radius: 10px;
    border-color: beige;
    font: bold 14px;
    min-width: 10em;
    padding: 6px;
}

```

/*imagesForExample: example_for_QPushButton3.png*/



```

4. QPushButton {
    background-color: red;
    border-style: outset;
    border-width: 2px;
    border-radius: 10px; /*边缘线的半径*/
    border-color: #fcf117; /*按钮的边缘线为黄色*/
    font: bold 14px; /*字体*/
    min-width: 10em;
    padding: 6px;
}

5. QPushButton:pressed {
    background-color: #7fb80e; /*当按钮被按下 (pressed) 的时候就出现若绿色按钮*/
    border-style: inset;
}

```

3.定制按钮(QPushButton)的 menu indicator 辅助控制器

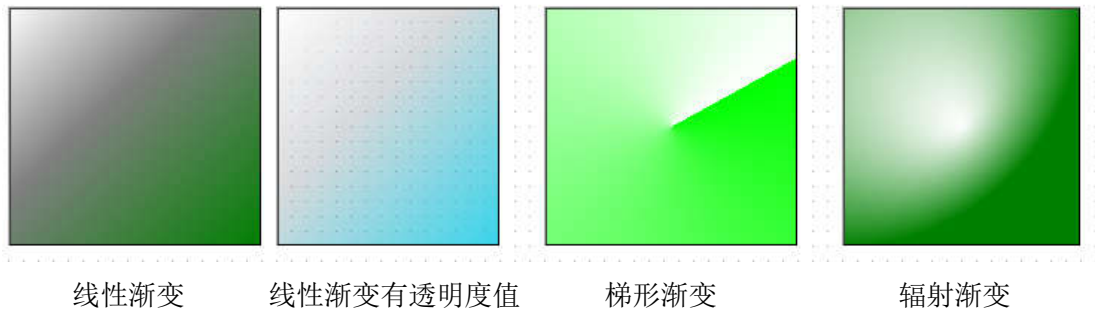
```
QPushButton::menu-indicator {  
    image: url(myindicator.png);  
    subcontrol-position: right center;  
    subcontrol-origin: padding;  
    left: -2px;  
}
```

4.指定某一个窗口部件进行设置属性

```
QPushButton#pushButton{  
    corlor:red;  
}  
/*指定的按钮 pushButton 前景色为红色*/
```

5. 渐变器

```
QTextEdit {  
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,  
                                stop:0 white, stop: 0.4 gray, stop:1 green)  
}  
  
/* linear gradient from white to green */  
QTextEdit {  
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1,  
                                stop:0 white, stop: 0.4 rgba(10, 20, 30, 40), stop:1 rgb(0, 200, 230, 200))  
}  
  
/* conical gradient from white to green */  
QTextEdit {  
    background: qconicalgradient(cx:0.5, cy:0.5, angle:30,  
                                stop:0 white, stop:1 #00FF00)  
}  
  
/* radial gradient from white to green */  
QTextEdit {  
    background: qradialgradient(cx:0, cy:0, radius: 1,  
                                fx:0.5, fy:0.5, stop:0 white, stop:1 green)  
}
```



6. 定制 QAbstractScrollArea

```
1. QTextEdit, QListView {
    background-color: #7fb80e;
    background-image: url(images/myindicator.png);
    background-attachment: scroll;
}
```

```
2. QTextEdit, QListView {
    background-color: red;
    background-image: url(images/myindicator.png);
    background-attachment: fixed;
}
```

/*1 与 2 的区别就在于背景图片是否依赖于滚动条, 1. scroll 是表示图片跟随滚动条一起滑动, 2. fixed 表示背景图片是不跟随滚动条滑动的。*/

7. 定制 QCheckBox

QCheckBox 是与 QRadioButton 的样式是同步的, 都是三态部件(1. 选中。2. 半选中[表示是一个 checkbox 中含一组 checkbox, 这一组有的被选中有的不被选中, 表现出的这种状态]。3. 未选中)

unchecked: 表示这个 QCheckBox 没有选中的时候对其设置效果。

Checked: 表示是在选中的时候 QCheckBox 对其设置效果。

```
1. QCheckBox {
    spacing: 5px;
}
```

/*可点击的小窗口和描述文字之间的空间为 5px*/

```
2. QCheckBox::indicator {
    width: 13px;
    height: 13px;
}
```

/*可选择小窗口的宽高设定*/

```

3. QCheckBox::indicator:unchecked {
    image: url(styles/start.png);
}
/*可点击的小窗口在不点击时添加的图片*/
/*imagesForExample: example_for_QCheckBox001.png*/

4. QCheckBox::indicator:unchecked:hover {
    image: url(styles/start.png);
}
/*鼠标划过 QCheckBox 时显示图片*/

5. QCheckBox::indicator:unchecked:pressed {
    image: url(styles/start.png);
}
/*单击文本描述的时候显示图片，但是小窗口显示的是”对号”符号*/

6. QCheckBox::indicator:checked {
    image: url(styles/start.png);
}
/*当已被选中的时候单击小窗口和文本描述的时候，小窗口变成这个图片*/

7. QCheckBox::indicator:checked:hover {
    image: url(styles/start.png);
}
/*当已被选中的时候鼠标划过 QCheckBox 时显示图片*/

8. QCheckBox::indicator:checked:pressed {
    image: url(styles/start.png);
}
/*当已被选中的时候鼠标点击 QCheckBox 显示图片，但是小窗口显示的是”对号”符号*/

9. QCheckBox::indicator:indeterminate:hover {
    image: url(styles/start.png);
}
/*不确定状态情况下的鼠标滑过（半选中状态）*/

10. QCheckBox::indicator:indeterminate:pressed {
    image: url(styles/start.png);
}
/*不确定状态情况下的鼠标滑过（半选中状态）*/

```

8. 定制 QComboBox

```

1. QComboBox {

```

```

border: 1px solid gray; /*外围框的 1px 的 gray 色的实线*/
border-radius: 3px; /*四个角半径的弧度 3px*/
padding: 5px 18px 5px 10px; /*中间文本与上、右、下、左之间的空隙*/
min-width: 20em; /*QComboBox 最小宽度*/
}

```

2. QComboBox:editable {


```

background: blue;
      
```

 }

/*可编辑的变为 blue*/

/*example_for_QComboBox001.png*/



3. QComboBox:!editable, QComboBox::drop-down:editable {


```

background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                             stop: 0 #f173ac, stop: 0.4 #00ae9d,
                             stop: 0.5 #596032, stop: 1.0 #ef4136);
      
```

 }

/*上到下的一种线性渐变器*/

/*imagesForExample : example_for_QComboBox002.png*/

4. /* QComboBox gets the "on" state when the popup is open */

/*不可编辑的 QComboBox 和 可编辑的 QComboBox 设置属性为 on*/

QComboBox:!editable:on, QComboBox::drop-down:editable:on {

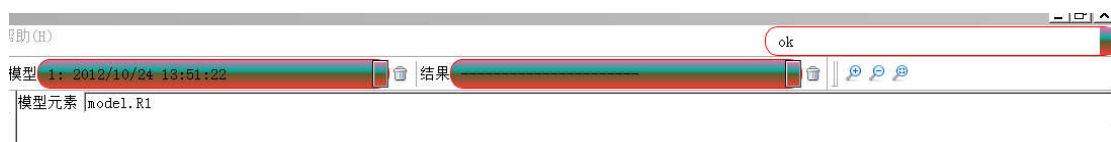

```

background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                             stop: 0 #f173ac, stop: 0.4 #00ae9d,
                             stop: 0.5 #596032, stop: 1.0 #ef4136)
        
```

 }

/*当点击展开按钮时的时候才可以出现*/

/*imagesForExample : example_for_QComboBox002.png 现象*/



5. QComboBox:on { /* shift the text when the popup opens */


```

padding-top: 3px;
padding-left: 4px;
      
```

 }

/*当点击展开的按钮时 (on), 文本与外框间的空隙统一设置 top 3px , left 4px*/


```

6. QComboBox::drop-down {
    subcontrol-origin: padding;
    subcontrol-position: top right;
    width: 15px;

    border-left-width: 1px;
    border-left-color: darkgray;
    border-left-style: solid; /* 一条实线 */
    border-top-right-radius: 3px; /* 不确定状态 */
    border-bottom-right-radius: 3px;
}

7. QComboBox::down-arrow {
    image: url(styles/start.png);
}
/*更换下拉按钮的向下的图标*/

8. QComboBox::down-arrow:on { /* shift the arrow when popup is open */
    top: 5px;
    left: 1px;
}

```

9. 定制 QDockWidget

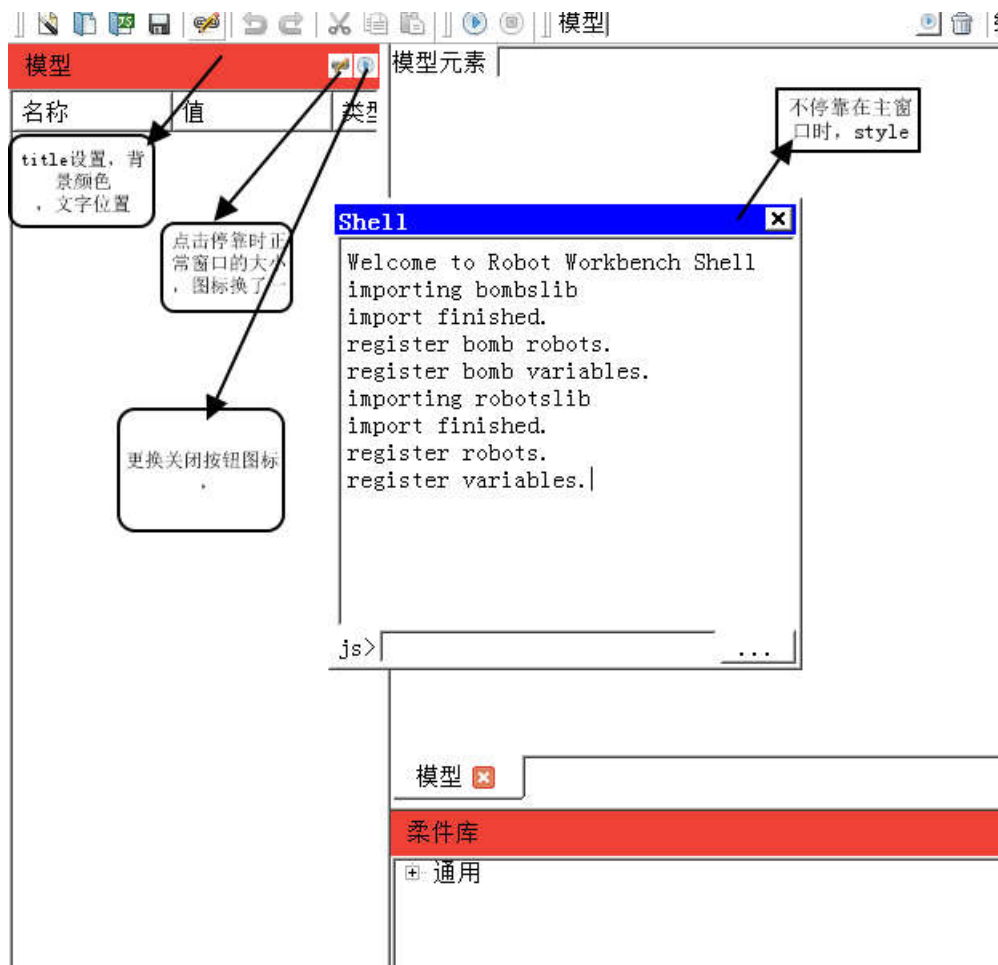
QDockWidget 的标题栏和按钮是可以自定义的。如下例题：

```

● 1. QDockWidget {
    border: 5px solid #ef4136; /*设置浮动窗口(QDockWidget)外框线*/
    titlebar-close-icon: url( styles/start.png);/*设置浮动窗口的关闭按钮*/
    titlebar-normal-icon: url( styles/linkedit.png);/*设置浮动窗口正常大小的按钮图标*/
}
/*imagesForExample : example_for_QDockWidget001.png 现象*/

2. QDockWidget::title {
    text-align: left; /* 浮动窗口的名字在左侧 */
    background: #ef4136; /*标题背景色*/
    padding-left: 10px; /*浮动窗口的名字与标题栏左侧外框距离*/
}
/*imagesForExample : example_for_QDockWidget001.png 现象*/

```



```
3. QDockWidget::close-button, QDockWidget::float-button {
    border: 1px solid transparent; /*两个按钮的外框线 1px, 实线, transparent (透明的)*/
    background: darkgray; /*按钮的背景颜色*/
    padding: 0px; /*两个按钮的里面的图标与图标外框的距离*/
}
```

```
4. QDockWidget::close-button:hover, QDockWidget::float-button:hover {
    background: gray;
    border: 1px solid transparent;
}
```

/*单独设置背景颜色时需要设置 border 颜色值才可以*/

```
5. QDockWidget::close-button:pressed, QDockWidget::float-button:pressed {
    padding: 1px -1px -1px 1px;
    border: 1px solid transparent;
}
```

/*同理也要设置 border 的值才可以出现效果*/

- 如果你想把 QDockWidget 的两个按钮放在 left 例子 6、7 如下:

```
1. QDockWidget {
    border: 1px solid lightgray;
```

```

        titlebar-close-icon: url(styles/start.png);
        titlebar-normal-icon: url(styles/linkedit.png);
    }

2.QDockWidget::title {
    text-align: left;
    background: lightgray;
    padding-left: 35px;
}

3.QDockWidget::close-button, QDockWidget::float-button {
    background: darkgray;
    padding: 0px;
    icon-size: 14px; /* maximum icon size 下面定义了 icon 的大小*/
}

4.QDockWidget::close-button:hover, QDockWidget::float-button:hover {
    background: gray;
}

5.QDockWidget::close-button:pressed, QDockWidget::float-button:pressed {
    padding: 1px -1px -1px 1px;
}

6.QDockWidget::close-button {
    subcontrol-position: top left;
    subcontrol-origin: margin;
    position: absolute;
    top: 0px; left: 0px; bottom: 0px;
    width: 14px;
}
/*上面是关闭按钮的设置*/

7.QDockWidget::float-button {
    subcontrol-position: top left;
    subcontrol-origin: margin; /*盒模型的最外面的那个控件选择默认是 padding 的,
此处设置为 margin*/
    position: absolute; /*绝对位置（它是相对于关闭按钮存在的）*/
    top: 0px; left: 16px; bottom: 0px;
    width: 14px;
}
/*上面是浮动按钮的设置*/

```

10. 定制 QFrame

一般用盒模式来设置 QFrame 的样式

```
1. QFrame, QLabel, QToolTip {
    border: 2px solid green;
    border-radius: 4px;
    padding: 2px;
    background-image: url(styles/start.png);
}
```

/*这个外框是 2px 实线，绿色，圆角半径是 4px 的，盒模型的 padding 是 2px，背景图片*/

11. 定制 QGroupBox

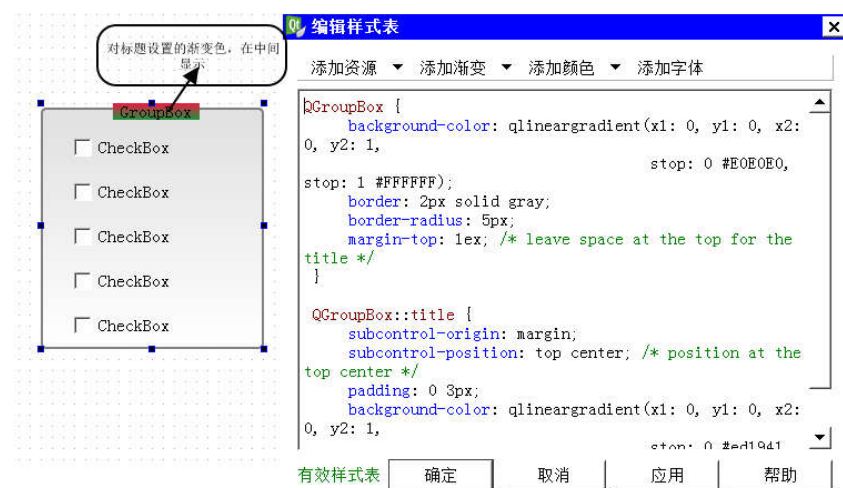
把 QGroupBox 的标题移到中间

```
QGroupBox {
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                      stop: 0 #E0E0E0, stop: 1 #FFFFFF);

    border: 2px solid gray;
    border-radius: 5px;
    margin-top: 1ex; /* leave space at the top for the title */
}
```

```
QGroupBox::title {
    subcontrol-origin: margin;
    subcontrol-position: top center; /* position at the top center */
    padding: 0 3px;
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                      stop: 0 #ed1941, stop: 1 #1d953f);
}
```

/*imagesForExample : example_for_QGroupBox001.png*/



可以像 QCheckBox 一样设置它的样式。

```
QGroupBox::indicator {
    width: 13px;
    height: 13px;
}

QGroupBox::indicator:unchecked {
    image: url(styles/start.png);
}
```

12. 定制 QHeaderView

```
QHeaderView::section {
    background-color: qlineargradient(x1:0, y1:0, x2:0, y2:1,
                                      stop:0 #616161, stop: 0.5 #505050,
                                      stop: 0.6 #434343, stop:1 #656565);

    /*标题头的渐变色*/
    color: #ef4136; /*字体的颜色*/
    padding-left: 4px; /*标题名字和左侧的位置*/
    border: 1px solid #6c6c6c; /*外框的线属性*/
}

QHeaderView::section:checked
{
    background-color: red; /*点击后标题头变为红色*/
}

QHeaderView::down-arrow {
    image: url(styles/start.png); /*图标设置用来排序的*/
}

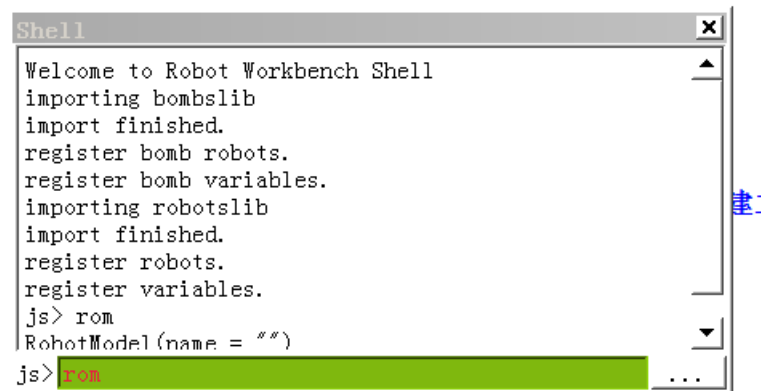
QHeaderView::up-arrow {
    image: url(styles/linkedit.png);
}
```

13 定制 QLineEdit

```
1. QLineEdit{
    background-color: #7fb80e;
    color: #fcf17;
}

/*前景（输入文字）颜色是#fcf17，背景颜色 #7fb80e;*/
/*imagesForExample : example_for_QLineEdit001.png*/
```

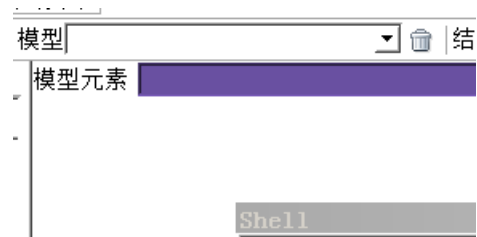
模型元素



```
2. QLineEdit[readOnly="true"] {
    color: #ea66a6;
    background-color: #6950a1;
}
```

/*只读(只可以读取不可以输入文字进行编辑的文字)对话框前景(文字)颜色是#ea66a6,背景颜色 #6950a1*/

/*imagesForExample : example_for_QLineEdit003.png*/



```
3. QLineEdit { color: red; }
```

```
    QLineEdit[readOnly="true"] { color: #fcab17; }
```

```
    QDialog QLineEdit { background-color: #7fb80e; }
```

/*在 QDialog 上的 QLineEdit 的背景颜色都为 #7fb80e*/

/*imagesForExample : example_for_QLineEdit002.png*/



14. 定制 QListView

设置表格交替的背景颜色可以这样设置。

```
QListView {
    alternate-background-color: yellow;
}
```

当鼠标划过表格的某一行的时候，我们可以如下的设置。

```
QListView {
    show-decoration-selected: 1; /* make the selection span the entire width of the
view */
}
```

```
QListView::item:alternate {
    background: #EEEEEE;
}
```

```
QListView::item:selected {
    border: 1px solid #6a6ea9;
}
/*选择时这一行被#6a6ea9 1px 的实线包围*/
```

```
QListView::item:selected:!active {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #ABAFE5, stop: 1 #8588B2);
}
/* 当点击一次表格，之后离开，单击后的这一行的颜色*/
```

```
QListView::item:selected:active {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #6a6ea9, stop: 1 #888dd9);
}
/*鼠标点击这一行时的背景颜色*/
```

```
QListView::item:hover {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #FAFBFE, stop: 1 #DCDEF1);
}
/*鼠标划过时背景颜色变化*/
```

15. 定制 QMainWindow

```
QMainWindow::separator {
    background: yellow;
```

```

        width: 10px; /* when vertical */
        height: 10px; /* when horizontal */
    }
/*主窗口的分割线为 yellow*/

QMainWindow::separator:hover {
    background: red;
}
/*当鼠标划过时为 red*/

```

16. 定制 QMenu

```

1. QMenu {
    background-color: #ABABAB; /* sets background of the menu */
    border: 1px solid black;
}
/*QMenu 的背景颜色, 这个 QMenu 的框为 1px 实线 的 black */
2. QMenu::item {
    /* sets background of menu item. set this to something non-transparent
       if you want menu color and menu item color to be different */
    background-color: transparent;
}
/*透明度*/

3. QMenu::item:selected { /* when user selects item using mouse or keyboard */
    background-color: #102b6a;
}
/*选择 item 时的背景颜色*/

4. QMenu { /* 菜单 */
    background-color: white;
    margin: 0px; /* some spacing around the menu */
    border: 1px solid green;
}

QMenu::item {
    padding: 2px 25px 2px 20px;
    border: 1px solid transparent; /* reserve space for selection border */
}

QMenu::item:selected {
    border-color: darkblue;
    background: rgba(100, 100, 100, 150);
}

```



```
QMenu::icon:checked { /* appearance of a 'checked' icon */
```

```
    background: red;
    border: 1px inset gray;
    position: absolute;
    top: 1px;
    right: 1px;
    bottom: 1px;
    left: 1px;
```

```
}
```

```
QMenu::separator {
```

```
    height: 2px;
    background: lightblue;
    margin-left: 10px;
    margin-right: 5px;
```

```
}
```

```
QMenu::indicator {
```

```
    width: 13px;
    height: 13px;
```

```
}
```

```
/* non-exclusive indicator = check box style indicator (see QActionGroup::setExclusive) */
```

```
QMenu::indicator:non-exclusive:unchecked {
```

```
    /*image: url(styles/start.png);*/
    /*image: url(styles/start.png);*/
```

```
}
```

```
QMenu::indicator:non-exclusive:unchecked:selected {
```

```
    /*image: url(styles/start.png);*/
    image: url(styles/start.png);
```

```
}
```

```
QMenu::indicator:non-exclusive:checked {
```

```
    /*image: url(styles/start.png);*/
    image: url(styles/start.png);
```

```
}
```

```
QMenu::indicator:non-exclusive:checked:selected {
```

```
    /*image: url(styles/start.png);*/
    image: url(styles/start.png);
```

```
}
```

```

/* exclusive indicator = radio button style indicator (see QActionGroup::setExclusive) */
QMenu::indicator:exclusive:unchecked {
    image: url(styles/start.png);
}

QMenu::indicator:exclusive:unchecked:selected {
    image: url(styles/start.png);
}

QMenu::indicator:exclusive:checked {
    image: url(styles/start.png);
}

QMenu::indicator:exclusive:checked:selected {
    image: url(styles/start.png));
}

```

17. 定制 QMenuBar

我们定制菜单栏时的用法，还有很多的用法多找找表格

```

QMenuBar { /* 菜单栏 */
    background-color: qlineargradient(x1:0, y1:0, x2:0, y2:1,
    stop:0 lightgray, stop:1 darkgray);
}

QMenuBar::item {
    spacing: 3px; /* spacing between menu bar items */
    padding: 3px 4px;
    background: transparent;
    border-radius: 4px;
}

QMenuBar::item:selected { /* when selected using mouse or keyboard */
    background: #a8a8a8;
}
/*选择时的背景颜色*/

QMenuBar::item:pressed {
    background: #1d953f;
}
/*点击鼠标时菜单栏的背景颜色*/

```

18. 定制 QProgressBar

```
QProgressBar {
    border: 2px solid grey;
    border-radius: 5px;
}
/*外框线是 2px*/

QProgressBar::chunk {
    background-color: #05B8CC;
    width: 20px;
}
/*背景颜色，和宽度*/

QProgressBar {
    border: 2px solid grey;
    border-radius: 5px;
    text-align: center;
}
/*文本在中间，圆角角度*/

QProgressBar::chunk {
    background-color: #CD96CD;
    width: 10px;
    margin: 0.5px;
}
/*结果就是一块一块的增加，间隔是 0.5px*/
```

19. 定制 QPushButton

[illegible]

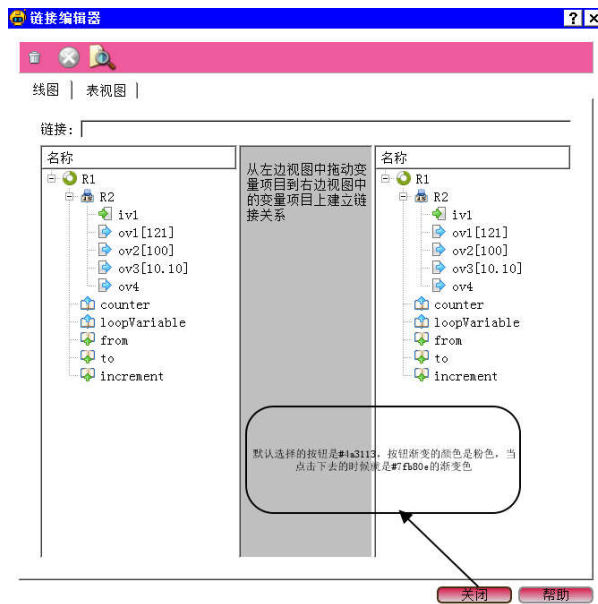
```

QPushButton:flat {
    border: none; /* no border for a flat push button */
}

QPushButton:default {
    border-color: #4a3113; /* make the default button prominent */
}

/*imagesFoeExample: example_for_QPushButton001.png*/

```



可以用:checked 状态来设置 QPushButton。
当把按钮放入菜单栏上时我们可以这样设置它的属性。

```

QPushButton:default {
    border-color: #4a3113; /* make the default button prominent */
}

QPushButton:open { /* when the button has its menu open */
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                      stop: 0 #1d953f, stop: 1 #f6f7fa);
}

QPushButton::menu-indicator {
    image: url(menu_indicator.png);
    subcontrol-origin: padding;
    subcontrol-position: bottom right;
}

QPushButton::menu-indicator:pressed, QPushButton::menu-indicator:open {
    position: relative;
}

```

```
    top: 2px; left: 2px; /* shift the arrow by 2 px */
}
```

20. 定制 QRadioButton

```
QRadioButton::indicator {
    width: 13px;
    height: 13px;
}
/*定制可选择的小圆圈的大小*/
QRadioButton::indicator::unchecked {
    image: url(/images/radiobutton_unchecked.png);
}

QRadioButton::indicator:unchecked:hover {
    image: url(/images/radiobutton_unchecked_hover.png);
}

QRadioButton::indicator:unchecked:pressed {
    image: url(/images/radiobutton_unchecked_pressed.png);
}

QRadioButton::indicator::checked {
    image: url(/images/radiobutton_checked.png);
}

QRadioButton::indicator:checked:hover {
    image: url(/images/radiobutton_checked_hover.png);
}

QRadioButton::indicator:checked:pressed {
    image: url(/images/radiobutton_checked_pressed.png);
}
/*小圆圈被选中和没被选中时的鼠标状态*/
```

21. 定制 QScrollBar

QScrollBar 这个滚动条可是通过下面的方式设置，也可用 handle, add-line, sub-line 一些辅助控制器设置。

Note: 如果它定义了一个属性或者是副属性，那么我们也要定义它的子部件的属性和副属性。

```
QScrollBar:horizontal {
    border: 2px solid grey;
```

```

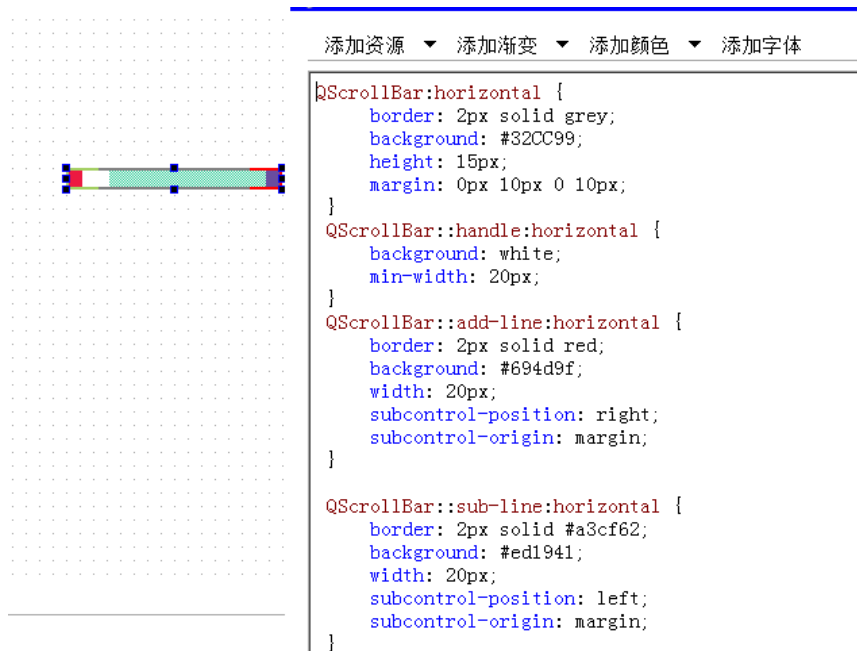
        background: #32CC99;
        height: 15px;
        margin: 0px 10px 0 10px; /*两头的是 10px 的小方块*/
    }
/*横向的滚动条外边缘线 2px 实线 grey 背景颜色是#32CC99, 高为 15px*/

QScrollBar::handle:horizontal {
    background: white;
    min-width: 20px;
}
/*滚动条的滑块 (handle) 为白色, 宽度为 20px*/

QScrollBar::add-line:horizontal {
    border: 2px solid grey;
    background: #694d9f;
    width: 20px;
    subcontrol-position: right;
    subcontrol-origin: margin;
}
/*向添加内容方向的小方块颜色, 宽度, 位置, 边缘线*/

QScrollBar::sub-line:horizontal {
    border: 2px solid grey;
    background: #ed1941;
    width: 20px;
    subcontrol-position: left;
    subcontrol-origin: margin;
}
/*向减少内容方向的小方块颜色, 宽度, 位置, 边缘线*/
/*imagesForExample: example_for_QScrollBar001.png*/

```



对于箭头方向的设置 left-arrow 和 right-arrow

```
QScrollBar:left-arrow:horizontal, QScrollBar::right-arrow:horizontal {
    border: 2px solid grey;
    width: 3px;
    height: 3px;
    background: white;
}
```

```
QScrollBar::add-page:horizontal, QScrollBar::sub-page:horizontal {
    background: none;
}
```

/*定制一个箭头的大小*/

这个是设置一个横向的加内容和减内容的的小方块放到了一起。

```
QScrollBar:horizontal {
    border: 2px solid green;
    background: cyan;
    height: 15px;
    margin: 0px 40px 0 0px;
}
```

```
QScrollBar::handle:horizontal {
    background: gray;
    min-width: 20px;
}
```

```
QScrollBar::add-line:horizontal {
```

```

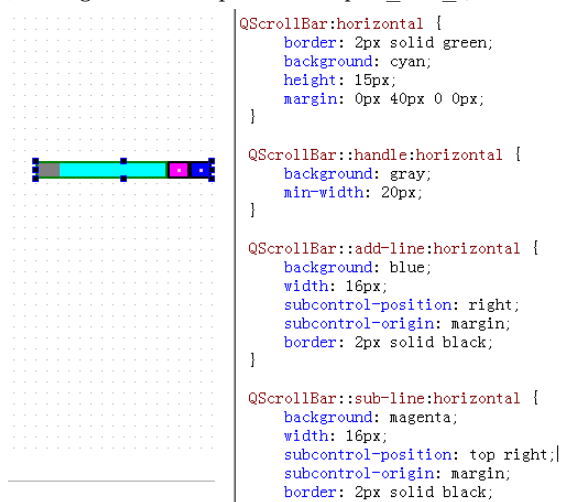
        background: blue;
        width: 16px;
        subcontrol-position: right;
        subcontrol-origin: margin;
        border: 2px solid black;
    }

    QScrollBar::sub-line:horizontal {
        background: magenta;
        width: 16px;
        subcontrol-position: top right;
        subcontrol-origin: margin;
        border: 2px solid black;
        position: absolute;
        right: 20px;
    }

    QScrollBar:left-arrow:horizontal, QScrollBar::right-arrow:horizontal {
        width: 3px;
        height: 3px;
        background: pink;
    }

    QScrollBar::add-page:horizontal, QScrollBar::sub-page:horizontal {
        background: none;
    }
/*imagesFoeExample: example_for_QScrollBar002.png*/

```



竖向的滚动条和纵向的滚动条设置时类似的。

```

QScrollBar:vertical {
    border: 2px solid grey;
    background: #32CC99;

```



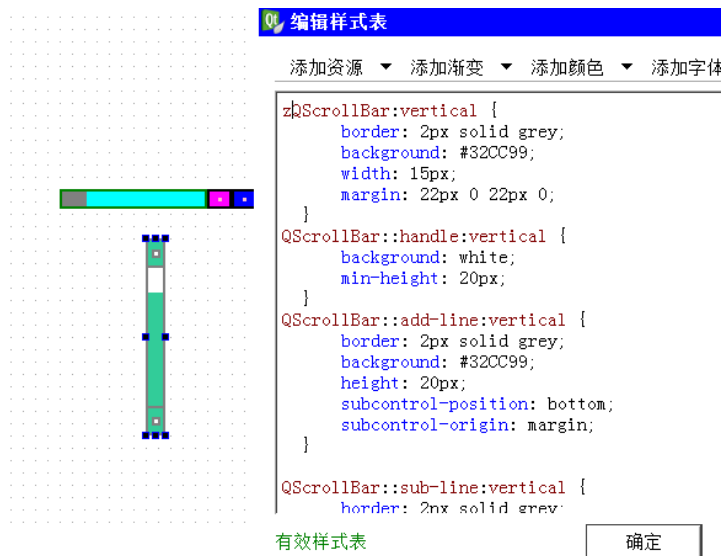
```

        width: 15px;
        margin: 22px 0 22px 0;
    }
    QScrollBar::handle:vertical {
        background: white;
        min-height: 20px;
    }
    QScrollBar::add-line:vertical {
        border: 2px solid grey;
        background: #32CC99;
        height: 20px;
        subcontrol-position: bottom;
        subcontrol-origin: margin;
    }

    QScrollBar::sub-line:vertical {
        border: 2px solid grey;
        background: #32CC99;
        height: 20px;
        subcontrol-position: top;
        subcontrol-origin: margin;
    }
    QScrollBar::up-arrow:vertical, QScrollBar::down-arrow:vertical {
        border: 2px solid grey;
        width: 3px;
        height: 3px;
        background: white;
    }

    QScrollBar::add-page:vertical, QScrollBar::sub-page:vertical {
        background: none;
    }
/*imagesForExample: example_for_QScrollBar003.png*/

```



22. 定制 QSizeGrip

仅仅是设置一张图片

```

QSizeGrip {
    image: url(styles/start.png));
    width: 16px;
    height: 16px;
}

```

23. 定制 QSlider

定制一个横向的窗口分割器

```

QSlider::groove:horizontal {
    border: 1px solid #ed1941;
    height: 8px;
    /* the groove expands to the size of the slider by default. by giving it a height,
    it has a fixed size */
    background: qlineargradient(x1:0, y1:0, x2:0, y2:1, stop:0 #494e8f, stop:1
#c4c4c4);
    margin: 2px 0;
}

```

/*QSlider 的槽部分设置*/

```

QSlider::handle:horizontal {
    background: qlineargradient(x1:0, y1:0, x2:1, y2:1, stop:0 #007d65, stop:1
#8f8f8f);
    border: 1px solid #5c5c5c;
    width: 18px;
}

```

```

        margin: -2px 0; /* handle is placed by default on the contents rect of the groove.
Expand outside the groove */
        border-radius: 3px;
    }
/*QSlider 的滑块设置*/
/* imagesForExample: example_for_QSplitter001.png*/

```



```

QSlider::groove:vertical {
    background: red;
    position: absolute; /* absolutely position 4px from the left and right of the
widget. setting margins on the widget should work too... */
    left: 4px; right: 4px;
}

```

```

QSlider::handle:vertical {
    height: 10px;
    background: green;
    margin: 0 -4px; /* expand outside the groove */
}

```

```

QSlider::add-page:vertical {
    background: white;
}

```

```

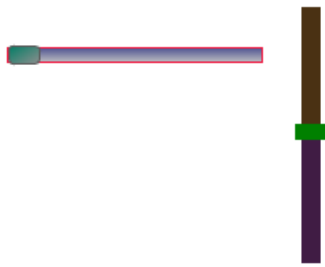
QSlider::sub-page:vertical {
    background: pink;
}

```

```

/* imagesForExample: example_for_QSplitter002.png*/

```



24. 定制 QSpinBox

```
QSpinBox {
    padding-right: 15px; /* make room for the arrows */
    border-image: url(styles/start.png) 4;
    border-width: 3;
}

QSpinBox::up-button {
    subcontrol-origin: border;
    subcontrol-position: top right; /* position at the top right corner */

    width: 16px; /* 16 + 2*1px border-width = 15px padding + 3px parent border */
    border-image: url(styles/start.png) 1;
    border-width: 1px;
}

QSpinBox::up-button:hover {
    border-image: url(styles/start.png) 1;
}

QSpinBox::up-button:pressed {
    border-image: url(styles/start.png) 1;
}

QSpinBox::up-arrow {
    image: url(styles/start.png);
    width: 7px;
    height: 7px;
}

QSpinBox::up-arrow:disabled, QSpinBox::up-arrow:off { /* off state when value is max */
```

```

        image: url(styles/start.png);
    }

    QSpinBox::down-button {
        subcontrol-origin: border;
        subcontrol-position: bottom right; /* position at bottom right corner */

        width: 16px;
        border-image: url(styles/start.png) 1;
        border-width: 1px;
        border-top-width: 0;
    }

    QSpinBox::down-button:hover {
        border-image: url(styles/start.png) 1;
    }

    QSpinBox::down-button:pressed {
        border-image: url(styles/start.png) 1;
    }

    QSpinBox::down-arrow {
        image: url(styles/start.png);
        width: 7px;
        height: 7px;
    }

    QSpinBox::down-arrow:disabled,
    QSpinBox::down-arrow:off { /* off state when value in min */
        image: url(styles/start.png);
    }

```

25. 定制 QSplitter

```

QSplitter::handle {
    image: url(styles/start.png);
}
/*句柄图片*/
QSplitter::handle:horizontal {
    width: 2px;
}

QSplitter::handle:vertical {
    height: 2px;
}

```

```

}

QSplitter::handle:pressed {
    image: url(styles/start.png));
}

```

26. 定制 QStatusBar

提供状态栏的背景颜色和状态栏里面 item 的外框线

```

1. QStatusBar {
    background: brown;
}

```

```

QStatusBar::item {
    border: 1px solid red;
    border-radius: 3px;
}

```

部件被加到状态栏上时这样设置才可以生效

```

2.QStatusBar QLabel {
    border: 3px solid white;
}

```

27. 定制 QTabWidget 和 QTabBar

```

QTabWidget::pane { /* The tab widget frame */
    border-top: 2px solid #C2C7CB;
}
/*外框线*/

```

```

QTabWidget::tab-bar {
    left: 5px; /* move to the right by 5px */
}
/*tab 页的按钮统一向右移 5px*/

```

```

/* Style the tab using the tab sub-control. Note that it reads QTabBar _not_ QTabWidget */
/*tab 一般用 QTabBar 控制如下*/

```

```

QTabBar::tab {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #E1E1E1, stop: 0.4 #DDDDDD,
                                stop: 0.5 #D8D8D8, stop: 1.0 #D3D3D3);
    border: 2px solid #C4C4C3;
}

```

```

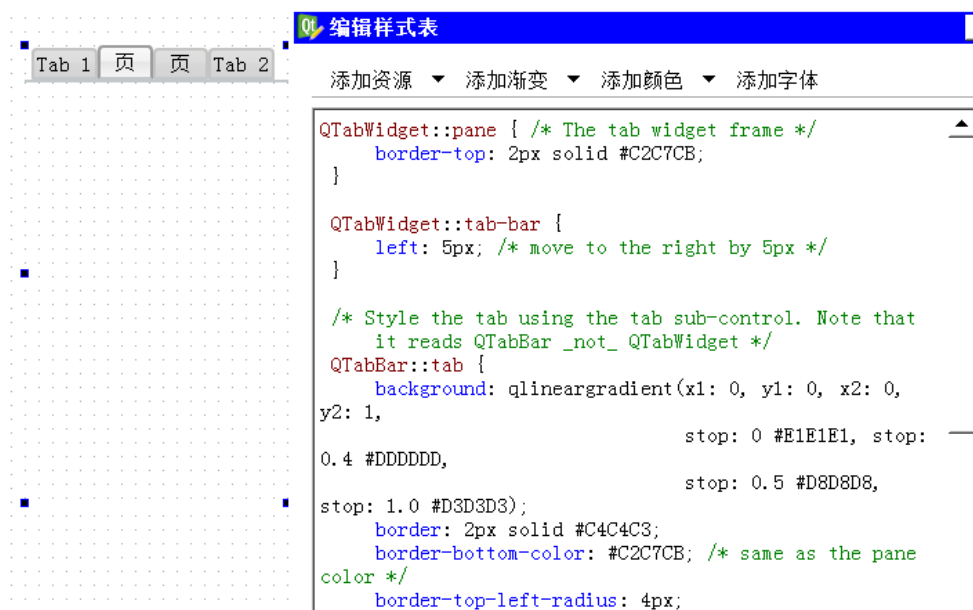
border-bottom-color: #C2C7CB; /* tab 的下线，pane 临近的线但是不会和 pane 冲突*/
border-top-left-radius: 4px; /*左上角的弧度*/
border-top-right-radius: 4px;
min-width: 8ex; /*最小的 tab*/
padding: 2px; /*可以理解为 tab 的高度*/
}
/*设置 tab，背景是一个渐变色，外框线 2px 实线 #C4C4C3，*/

QTabBar::tab:selected, QTabBar::tab:hover {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #fafafa, stop: 0.4 #f4f4f4,
                                stop: 0.5 #e7e7e7, stop: 1.0 #fafafa);
}
/*鼠标划过和鼠标选择时的背景颜色*/

QTabBar::tab:selected {
    border-color: #9B9B9B;
    border-bottom-color: #C2C7CB; /* same as pane color */
}
/*鼠标单击后 tab 的边缘线颜色和下边缘线的颜色*/

QTabBar::tab:!selected {
    margin-top: 2px; /* make non-selected tabs look smaller */
}
/*没被选择时 tab 整体下降 2px，目的凸显被选中的 tab*/
/*imagesFoeExample: example_for_QTabWidget001.png*/

```



上面的 tab 是酷似一个按钮一个按钮的，但是有时候我们想要一种效果就是，当我们选择 tab 的时候这个 tab 凸现出来，并放大，如下一个例子：

```

QTabWidget::pane { /* The tab widget frame */
    border-top: 2px solid #C2C7CB;
}
/*tabwidget 的上面线的颜色*/

QTabWidget::tab-bar {
    left: 5px; /* move to the right by 5px */
}
/*整体的 tab 键左移 5px*/

QTabBar::tab {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #E1E1E1, stop: 0.4 #DDDDDD,
                                stop: 0.5 #D8D8D8, stop: 1.0 #D3D3D3);

    border: 2px solid #C4C4C3;
    border-bottom-color: #C2C7CB; /* same as the pane color */
    border-top-left-radius: 4px;
    border-top-right-radius: 4px;
    min-width: 8ex;
    padding: 2px;
}

QTabBar::tab:selected, QTabBar::tab:hover {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #fafafa, stop: 0.4 #f4f4f4,
                                stop: 0.5 #e7e7e7, stop: 1.0 #fafafa);
}

QTabBar::tab:selected {
    border-color: #9B9B9B;
    border-bottom-color: #C2C7CB; /* same as pane color */
}
/*选择时的线的颜色*/

QTabBar::tab:!selected {
    margin-top: 2px; /* make non-selected tabs look smaller */
}
/*不被选择时下移 2px*/

/* make use of negative margins for overlapping tabs */
QTabBar::tab:selected {
    /* expand/overlap to the left and right by 4px */
    margin-left: -4px;
}

```



```

        margin-right: -4px;
    }
    /*当被点击的时候就会向外左右扩展 4px*/

    QTabBar::tab:first:selected {
        margin-left: 0; /* the first selected tab has nothing to overlap with on the left */
    }
    /*意思是说你选择第一个 tab 它的左面是不增加的*/

    QTabBar::tab:last:selected {
        margin-right: 0; /* the last selected tab has nothing to overlap with on the right */
    }
    /*选择最后一个的时候它的右侧是不增加的*/

    QTabBar::tab:only-one {
        margin: 0; /* if there is only one tab, we don't want overlapping margins */
    }
    /*只有一个的时候是不扩展的*/

```

3. 有时候我们想要一个可以把 tab 放在 pane 线上。例子如下:

```

QTabWidget::pane { /* The tab widget frame */
    border-top: 2px solid #C2C7CB;
    position: absolute;
    top: -0.5em;
}
/*widget 的上线颜色为#C2C7CB，位置是完全绝对与原来的位置，-0.5 是在原来（原来是在
tab 下线的下面）的位置上面 0.5em 的距离。*/

QTabWidget::tab-bar {
    alignment: center;
}
/*所有的 tab 都放在中间的位置*/

QTabBar::tab {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #E1E1E1, stop: 0.4 #DDDDDD,
                                stop: 0.5 #D8D8D8, stop: 1.0 #D3D3D3);

    border: 2px solid #C4C4C3;
    border-bottom-color: #C2C7CB; /* same as the pane color */
    border-top-left-radius: 4px;
    border-top-right-radius: 4px;
    min-width: 8ex;
    padding: 2px;
}

```

```

    }
/*对 tab 设置颜色，外框*/
QTabBar::tab:selected, QTabBar::tab:hover {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                stop: 0 #fafafa, stop: 0.4 #f4f4f4,
                                stop: 0.5 #e7e7e7, stop: 1.0 #fafafa);
}
/*tab 键对鼠标操作的反应*/

QTabBar::tab:selected {
    border-color: #9B9B9B;
    border-bottom-color: #C2C7CB; /* same as pane color */
}

```

下面是对 **tear** 设置的，当 **tab** 页过多页面有限显示不出所有的 **tab**，系统一般会自动为我们添加一个 **scroller** 来向左向右操作 **tab** 键，以至于你可以操作所有的 **tab**。以下是对 **tab** 的 **scroller** 设置的样式代码：

```

QTabBar::tear {
    image: url(styles/linkedit.png);
}

QTabBar::scroller { /* the width of the scroll buttons */
    width: 20px;
}
/*scroller 是两个按钮的形式，它的宽度为 20px*/

QTabBar QToolButton { /* the scroll buttons are tool buttons */
    border-image: url(styles/start.png) 2;
    border-width: 2px;
}
/*给 buttons 添加图片，九宫格分割法分割图片离四个角的距离是 2px*/
/*按钮的外框的线粗*/

QTabBar QToolButton::right-arrow { /* the arrow mark in the tool buttons */
    image: url(styles/start.png);
}
/*右按钮*/

QTabBar QToolButton::left-arrow {
    image: url(styles/start.png);
}
/*左按钮*/
当有同样的设置时比如
QTabWidget::tab-bar {

```

```

        alignment: center;
    }
    /*居中*/
    QTabWidget::tab-bar {
        left: 5px; /* move to the right by 5px */
    }
    /*左起 5px*/同时设置就会以 left: 5px 为准*/

```

28. 定制 QTableView

这是一个表格，当你选择某一个表格的时候，你会设置这个表格的颜色，如下：

```

QTableView {
    selection-background-color: qlineargradient(x1: 0, y1: 0, x2: 0.5, y2: 0.5,
                                                stop: 0 #FF92BB, stop: 1 white);
}
/*selection-background-color 选择的背景颜色*/

QTableView QTableCornerButton::section {
    background: red;
    border: 2px outset red;
}
/*表头的重合的部分的颜色变化*/

```

29. 定制 QToolBar

工具栏的设定，

```

QToolBar {
    background: red;
    spacing: 3px; /* spacing between items in the tool bar */
}
/*工具栏的背景颜色*/

QToolBar::handle {
    image: url(styles/start.png);
}
/*工具栏可移动的 handle 的图标*/

```

30. 定制 QToolBox

这是如同 QQ 好友类型的部件。如下例子：

```

QToolBox::tab {
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,

```

```

                                stop: 0 #E1E1E1, stop: 0.4 #DDDDDD,
                                stop: 0.5 #7fb80e, stop: 1.0 #D3D3D3);

    border-radius: 5px;
    color: darkgray;
}
/*按钮的背景色是渐变的，角弧度 5px，字体的颜色*/

QToolBox::tab:selected { /* italicize selected tabs */
    font: italic;
    color: white;
}
/*选中一个的时候它的字体变化了，字的颜色变为 white*/

```

31. 定制 QPushButton

QPushButton 有三种形式的：

- 1.当 QPushButton 没有在 menu 上时他可以像定制，QPushButton 一样定制它；
- 2.当 QPushButton 在 menu 上，并且他的 QPushButton::popupMode 设置为 QPushButton::DelayedPopup 或者 QPushButton::instantPopup,就可以像定制 QPushButton 在菜单栏里用 menu-indicator 定制样式；
- 3.当 QPushButton 的 QPushButton::popupMode 设置为 QPushButton::MenuButtonPopup,在 这种情况下例子如下：

```

QPushButton { /* all types of tool button */
    border: 2px solid #8f8f91;
    border-radius: 6px;
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                      stop: 0 #f6f7fa, stop: 1 #dadbde);
}
/*所有的 toolbutton 的外框颜色，背景颜色，和角弧度*/

QPushButton[popupMode="1"] { /* only for MenuButtonPopup */
    padding-right: 20px; /* make way for the popup button */
}
/*当设置为 popupMode 的性质为真的时候，（只对 MenuButtonPopup 按钮有效）他的右侧
填充为 20px*/

```

```

QPushButton:pressed {
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,
                                      stop: 0 #red, stop: 1 #f6f7fa);
}
/*点击时背景颜色*/

```

```

QPushButton:hover{
    background-color: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1,

```

```

stop: 0 blue, stop: 1 #f6f7fa);
}
/*鼠标划过时的背景颜色*/

/* the subcontrols below are used only in the MenuButtonPopup mode */
/*下面的只适合 MenuButtonPopup 模式*/
/*以下对本软件无效可以猜测本软件用的不是 MenuButtonPopup 模式*/
QToolButton::menu-button {
    border: 2px solid gray;
    border-top-right-radius: 6px;
    border-bottom-right-radius: 6px;
    /* 16px width + 4px for border = 20px allocated above */
    width: 16px;
}

QToolButton::menu-arrow {
    image: url(styles/start.png);
}
/*箭头图片*/

QToolButton::menu-arrow:open {
    top: 1px; left: 1px; /* shift it a bit */
}

```

32. 定制 QToolTip

工具栏提示工具类似于 QLabel，可以设置它的不同明度，默认的是跟平台有关的一个部件，可以如下定义：

```

QToolTip {
    border: 2px solid darkkhaki; /*外框颜色*/
    background-color: #fdb933; /*背景颜色*/
    color: red; /*字体的颜色*/
    padding: 1px;
    border-radius: 3px;
    opacity: 200; /*透明度*/
}
/*Warning：提示 ToolTip 一定要在 QToolButton 样式表之前进行定义。*/

```

33. 定制 QTreeView

定制背景颜色交替

```

QTreeView {
    alternate-background-color: yellow;
}

```

```
}
```

我们可以定制特别的颜色，比如鼠标划过时 item 的颜色，用 item 辅助控制器选择：

```
QTreeView {  
    show-decoration-selected: 1;  
}
```

```
QTreeView::item {  
    border: 1px solid #d9d9d9;  
    border-top-color: transparent;  
    border-bottom-color: red;  
}
```

/*一个 item 不同内容之间的分割线，每一个内容有四个方向的 border，此时表示左右为 #d9d9d9，下面的颜色为 red，top 方向无颜色*/

```
QTreeView::item:hover {  
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #e7effd, stop: 1 #cbdaf1);  
    border: 1px solid blue  
}
```

/*当鼠标划过时 item 的背景颜色，边缘显示 blue*/

```
QTreeView::item:selected {  
    border: 1px solid #567dbc;  
}
```

/*选择时外框颜色的颜色*/

```
QTreeView::item:selected:active{  
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #6ea1f1, stop: 1 #567dbc);  
}
```

/*选中当前活动的 item 的背景色*/

```
QTreeView::item:selected:!active {  
    background: qlineargradient(x1: 0, y1: 0, x2: 0, y2: 1, stop: 0 #6b9be8, stop: 1 #577fbf);  
}
```

/*当选中了一个 item 时候，之后鼠标离开时点击其他非此 QTreeView 范围内时背景变的颜色*/

用::branch 辅助控制器来控制单独一个分支，如下：

```
QTreeView::branch {  
    background: palette(base);  
}
```

```
QTreeView::branch:has-siblings:!adjoins-item {  
    background: #130c0e;
```

```
}
```

/*是指 tree 打开后没有和后面链接的地方*/

```
QTreeView::branch:has-siblings:adjoins-item {  
    background: red;  
}
```

```
QTreeView::branch:!has-children:!has-siblings:adjoins-item {  
    background: blue;  
}
```

/*没有子目录，没有兄弟目录，有链接*/

```
QTreeView::branch:closed:has-children:has-siblings {  
    background: pink;  
}
```

/*有子目录。有兄弟目录的关闭时显示 pink 色*/

```
QTreeView::branch:has-children:!has-siblings:closed {  
    background: gray;  
}
```

/*有子目录，没有兄弟目录时是 gray*/

```
QTreeView::branch:open:has-children:has-siblings {  
    background: magenta;  
}
```

/*有子目录。有兄弟目录的打开时显示 pink 颜色*/

```
QTreeView::branch:open:has-children:!has-siblings {  
    background: green;  
}
```

/*有子目录没有兄弟目录的是 green*/

模型

✕

名称	值	类型	描述
R1			
R2			
	0	Double	X轴
	<edit. ...	位置点...	位置点变量
	<edit. ...	索引点...	索引点变量
	<edit. ...	Double[]	三个元...
	0	Double	Sin函数
c...	0	Refere...	
l...	model. ...	Refere...	
f...	0.1	Refere...	
to	0.9	Refere...	
i...	0.01	Refere...	

以上讲的是各种颜色，但是更实用的是使用各种图片为我么打造更好看的 QTreeView:



```
QTreeView::branch:has-siblings:!adjoins-item {  
    border-image: url(vline.png) 0;  
}
```

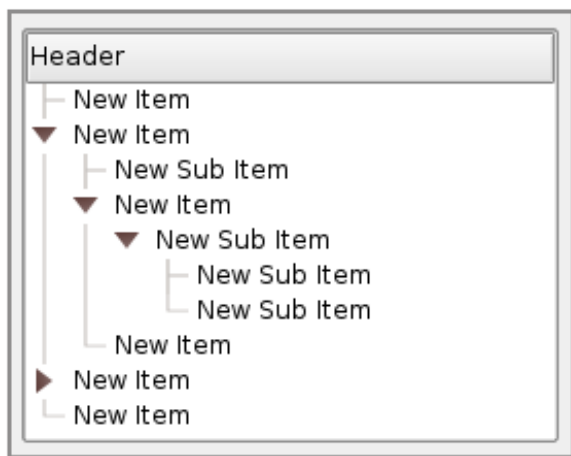
```
QTreeView::branch:has-siblings:adjoins-item {  
    border-image: url(branch-more.png) 0;  
}
```

```
QTreeView::branch:!has-children:!has-siblings:adjoins-item {  
    border-image: url(branch-end.png) 0;  
}
```

```
QTreeView::branch:has-children:!has-siblings:closed,  
QTreeView::branch:closed:has-children:has-siblings {  
    border-image: none;  
    image: url(branch-closed.png);  
}
```

```
QTreeView::branch:open:has-children:!has-siblings,  
QTreeView::branch:open:has-children:has-siblings {  
    border-image: none;  
    image: url(branch-open.png);  
}
```

最后的图形效果是:



34. QPushButton 和 images

这里我们讲一些关于 QPushButton 的样式设置的问题，当我们对他进行操作的时候，对其设置属性时我们往往用 **background-image** 属性设置，但是这个属性是一个很有缺陷的，例如，**background** 经常会提供隐藏在后面的 **decoration**，

那么什么是 **decoration**？

如下：

```
QTreeView {  
    show-decoration-selected: 1;  
}
```

我们在例子中经常会出现这个样式属性设置。

所以 **background** 这个属性是不成熟的，考虑不周全的，如果按钮被重新设置大小，那么背景是被拉伸或者平铺，这样看起来就不那么美观了。

最好用 **border-image** 这个属性，他可以很好地解决这个问题。

如下参考：

```
QPushButton {  
    color: grey;  
    border-image: url(I:/aboutstylesheet/imagesForExample/example_for 001.png) 3 10 3 10;  
    border-top: 3px transparent;  
    border-bottom: 3px transparent;  
    border-right: 10px transparent;  
    border-left: 10px transparent;  
}
```

这是一个九宫格分割法，它避免了 **background-image** 这个属性的拉伸图片的处理方式，**border-image** 把图片分成九个格子，顾为九宫格分割法。

七、参考网站

<qthelp://com.trolltech.qt.482/qdoc/stylesheet.html#overview>

<http://doc.qt.nokia.com/4.7/stylesheet.html>

<http://doc.qt.nokia.com/4.7/stylesheet-syntax.html>