第1章　使用 Qt Designer 定制观感

Qt 6 能够通过大多数人熟悉的方法轻松设计程序的用户界面。Qt 不仅提供了一个强大的用户界面（UI）工具包，称为 Qt Designer，它使我们能够在不编写一行代码的情况下设计用户界面，而且还允许高级用户通过一种称为 Qt 样式表的简单脚本语言来定制他们的用户界面组件。

本章主要涉及下列主题。

- 在 Qt Designer 中使用样式表。
- 定制基本样式表。
- 使用样式表创建登录界面。
- 在样式表中使用资源。
- 定制属性和子组件。
- 在 Qt 建模语言中进行样式设计。
- 将 QML 对象指针暴露给 C++。

1.1　技术要求

本章需要使用 Qt 6.1.1 MinGW 64-bit 和 Qt Creator 12.0.2。本章中使用的代码可以从本书的 GitHub 仓库下载，对应网址为 https://github.com/PacktPublishing/QT6-C-GUI-Programming-Cookbook---Third-Edition-/tree/main/Chapter01。

1.2　在 Qt Designer 中使用样式表

在这个示例中，我们将学习如何通过使用样式表和资源改变程序的观感，使其看起来更加专业。Qt 允许使用一种称为 Qt 样式表的样式表语言来装饰图形用户界面（GUI），这与网页设计师装饰网站的层叠样式表（CSS）非常相似。

1.2.1　实现方式

本节开始学习如何创建一个新项目，并熟悉 Qt Designer：

（1）打开 Qt Creator 并创建一个新项目。如果是第一次使用 Qt Creator，可以单击 Create Project…按钮，或者从顶部菜单栏选择 File | New Project…。

（2）在 Projects 窗口中选择 Application (Qt)并选择 Qt Widgets Application。

（3）单击底部的 Choose...按钮。随后会弹出一个窗口，要求输入项目的名称和位置。

（4）多次单击 Next 按钮，然后单击 Finish 按钮创建项目。目前我们使用默认设置。项目创建完成后，首先看到的是窗口左侧带有大量大图标的面板，这称为模式选择器（Mocle Selector）面板。

（5）我们会在侧边栏面板上看到所有源文件的列表，该面板位于模式选择器面板旁边。这里，可以选择要编辑的文件。在这种情况下，这是 mainwindow.ui 文件，并开始设计程序的 UI。

（6）双击 mainwindow.ui 文件，我们会看到一个完全不同的界面。Qt Creator 帮助我们从脚本编辑器切换到 UI 编辑器（Qt Designer），因为它检测到尝试打开的文件扩展名为 .ui。

（7）模式选择器面板上高亮的按钮已经从 Edit 变为 Design。我们可以通过单击模式选择器面板上半部分的任一按钮，切换回脚本编辑器或更改为其他工具。

（8）返回至 Qt Designer 并查看 mainwindow.ui 文件。这是程序的主窗口（如文件名所示），默认情况下它是空的，且没有任何组件。我们可以尝试通过按下模式选择器面板底部的 Run 按钮（绿色箭头按钮）编译和运行程序；编译完成后，会看到一个空窗口弹出。

（9）通过单击 Widget Box 区域（在 Buttons 类别下）中的 Push Button 项，将其拖曳至表单编辑器中的主窗口上，为我们程序的 UI 添加一个推送按钮。保持推送按钮的选择状态；我们将在窗口右侧的属性编辑器（Property Editor）区域看到此按钮的所有属性。向下滚动到中间，寻找一个名为 styleSheet 的属性。这是将为组件应用样式的地方，这些样式可能会从其子组件递归继承，具体取决于如何设置样式表。或者，也可以在表单编辑器中的任何组件上单击右键，并从弹出菜单中选择 Change styleSheet...。

（10）可以直接在 styleSheet 属性的输入框中编写样式表代码，或者单击输入框旁边的…按钮，打开 Edit Style Sheet 窗口，该窗口提供了更大的空间以便于编写更长的样式表代码。在窗口顶部，我们可以找到几个按钮，如 Add Resource、Add Gradient、Add Color 和 Add Font，如果记不住属性名称，这些按钮可以帮助我们开始编码。让我们尝试使用 Edit Style Sheet 窗口进行一些简单的样式设计。

（11）单击 Add Color 并选择一种颜色。

（12）在颜色选择器窗口中随机挑选一种颜色，如纯红色。然后，单击 OK 按钮。

（13）Edit Style Sheet 窗口的文本框中已添加了一行代码，如下所示。

```
color: rgb(255, 0, 0);
```

（14）单击 OK 按钮，推送按钮上的文本应该会变成红色。

1.2.2　工作方式

在开始学习如何设计自己的用户界面之前，让我们花点时间熟悉一下 Qt Designer 的界面，如图 1.1 所示。

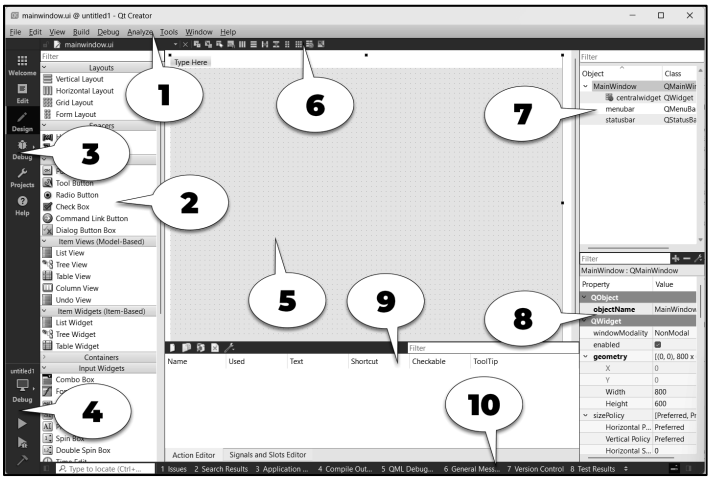

图 1.1　Qt Designer 的界面

图 1.1 的解释如下所示。

（1）菜单栏（Menu Bar）：这里包含特定于应用程序的菜单，提供对基本功能的便捷访问，如创建新项目、保存文件、撤销、重做、复制和粘贴。它还允许访问 Qt Creator 提供的开发工具，如编译器、调试器和分析器。

（2）Widget Box：这里可以找到 Qt Designer 提供的所有不同类型的组件。可以通过单击 Widget Box 区域中的一个组件并将其拖曳至表单编辑器中，将组件添加到程序的用户界面。

（3）模式选择器（Mode Selector）：模式选择器是一个侧边面板，提供不同工具的快捷按钮以便于访问。可以通过单击模式选择器面板上的 Edit 或 Design 按钮，快速在脚本编辑器和表单编辑器之间切换，这对于多任务处理非常有用。我们也可以同样快速方便地导航到调试器和分析器工具。

（4）构建快捷方式（Build Shortcuts）：构建快捷方式位于模式选择器面板的底部。可以通过单击这里的快捷按钮轻松构建、运行和调试项目。

（5）表单编辑器（Form Editor）：表单编辑器是编辑程序用户界面的地方。可以通过从 Widget Box 区域选择一个组件并将其拖曳至表单编辑器中，进而为程序添加不同的组件。

（6）表单工具栏（Form Toolbar）：这里可以快速选择不同的表单进行编辑。单击位于 Widget Box 区域顶部的下拉框，并选择想要用 Qt Designer 打开的文件。下拉框旁边的按钮可以在表单编辑器的不同模式之间切换，此外还有按钮可以更改用户界面布局。

（7）对象检查器（Object Inspector）：对象检查器区域列出了当前 .ui 文件中的所有组件。所有组件都根据它们在层级结构中的父子关系进行排列。可以选择对象检查器区域中的一个组件，在属性编辑器区域显示其属性。

（8）属性编辑器（Property Editor）：这里将显示从对象检查器区域或表单编辑器窗口中选择的组件的所有属性。

（9）Action Editor 和 Signals & Slots Editor：这个窗口包含 Action Editor 和 Signals & Slots Editor 两个编辑器。二者都可以通过窗口下方的标签页访问。Action Editor 是创建可以添加到程序用户界面的菜单栏或工具栏中的动作的地方。

（10）输出窗格（Output Panes）：输出窗格由几个不同的窗口组成，显示与脚本编译和调试相关的信息和输出消息。可以通过单击前面带有数字的按钮在不同的输出窗格之间切换，如 1 Issues、2 Search Results 或 3 Application Output。

1.2.3　附加内容

当前案例讨论了如何通过 C++编码将样式表应用到 Qt 组件上。尽管这种方法效果很好，但大多数时候，负责设计程序用户界面的人并不是程序员，而是专门设计用户友好界面的 UI 设计师。在这种情况下，最好让 UI 设计师使用不同的工具设计程序的布局和样式表，而不是去干预代码。Qt 提供了一个名为 Qt Creator 的一体化编辑器。

Qt Creator 包含几种不同的工具，如脚本编辑器、编译器、调试器、分析器和用户界面编辑器。用户界面编辑器，也称为 Qt Designer，是设计师在不编写任何代码的情况下设计程序用户界面的理想工具。这是因为 Qt Designer 采用了所见即所得的方法，提供了最终结果的准确视觉表现，这意味着用 Qt Designer 设计的任何内容，在程序编译和运行时视觉上都会呈现出相同的效果。

Qt 样式表与 CSS 之间的相似之处如下所示。

- 这是典型的 CSS 代码样式。

```
h1 { color: red; background-color: white; }
```

- 这是 Qt 样式表的样子，与上述 CSS 几乎相同。

```
QLineEdit { color: red; background-color: white; }
```

可以看到，它们都包含一个选择器和一个声明块。每个声明包含一个属性和一个值，并用冒号分隔。在 Qt 中，可以通过在 C++代码中调用 QObject::setStyleSheet()函数，将样式表应用到单个组件上。

例如，考虑以下情况：

```
myPushButton->setStyleSheet("color : blue");
```

上述代码会将名为 myPushButton 变量的按钮文本颜色更改为蓝色。此外，也可以通过在 Qt Designer 中的样式表属性字段编写声明来实现相同的结果。我们将在 1.3 节中更详细地讨论 Qt Designer。

Qt 样式表还支持 CSS2 标准中定义的所有不同类型的选择器，包括通用选择器、类型选择器、类选择器和 ID 选择器，这允许将样式应用到非常特定的单个组件或组件组。例如，如果想要更改具有 usernameEdit 对象名的特定行编辑组件的背景颜色，可以通过使用 ID 选择器来引用它。

```
QLineEdit#usernameEdit { background-color: blue }
```

注意：

要了解 CSS2 中所有可用的选择器（这些选择器也受到 Qt 样式表的支持），请参阅以下文档：http://www.w3.org/TR/REC-CSS2/selector.html。

1.3　定制基本样式表

在之前的案例中，我们学习了如何使用 Qt Designer 为组件应用样式表。这一次，让我们更进一步，创建一些其他类型的组件，并将它们的样式属性更改为一些奇特的样式。

然而，我们不会逐个为每个组件应用样式；相反，我们将学习将样式表应用到主窗口，并让它沿层级结构继承到所有其他组件，以便在长期运行中更容易管理和维护样式表。

1.3.1　实现方式

在以下示例中，我们将在画布上格式化不同类型的组件，并在样式表中添加一些代码来改变其外观：

（1）通过选择 PushButton 并单击 styleSheet 属性旁边的小箭头按钮，从 PushButton 中移除样式表。该按钮会将属性恢复到其默认值，在这种情况下是空的样式表。

（2）通过从 Widget Box 区域逐个拖曳至表单编辑器中，向 UI 添加更多组件。此处添加了一个行编辑框、一个组合框、一个水平滑块、一个单选按钮和一个复选框。

（3）为了简单起见，通过在对象检查器区域选择 menuBar、mainToolBar 和 statusBar，然后单击右键并选择 Remove，进而从 UI 中删除它们。现在，UI 应如图 1.2 所示。

（4）从表单编辑器或对象检查器区域选择主窗口，然后单击右键并选择 Change styleSheet...以打开 Edit Style Sheet 窗口。将以下内容插入到样式表中：

```
border: 2px solid gray;
border-radius: 10px;
padding: 0 8px;
background: yellow;
```

（5）我们将看到一个外观奇特的用户界面，所有内容都被黄色覆盖并带有粗边框。这是因为前述样式表没有选择器，这意味着样式将沿层级结构向下应用到主窗口的子组件。为了改变这一点，让我们尝试一些不同的方法。

```
QPushButton {
      border: 2px solid gray;
      border-radius: 10px;
      padding: 0 8px;
      background: yellow;
}
```

（6）这一次，只有 PushButton 将获得前面代码中描述的样式，所有其他组件将恢复为默认样式。我们可以尝试向用户界面添加更多按钮，它们都将具有相同的外观，如图 1.3 所示。

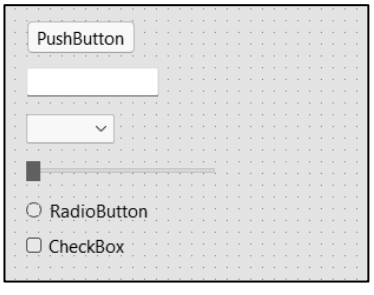

图 1.2　将一些组件拖曳至表单编辑器上

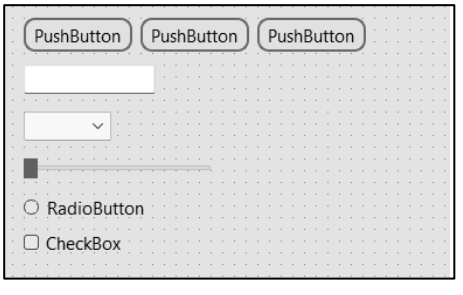

图 1.3　将 PushButton 按钮更改为黄色

（7）这是因为我们明确指示选择器将样式应用到所有 QPushButton 类的组件上。此外，

也可以通过在样式表中提及其名称，仅将样式应用到其中一个按钮上，对应代码如下所示。

```
QPushButton#pushButton_3 {
      border: 2px solid gray;
      border-radius: 10px;
      padding: 0 8px;
      background: yellow;
}
```

（8）一旦理解了这种方法，我们可以将以下代码添加到样式表中。

```
QPushButton {
      color: red;
      border: 0px;
      padding: 0 8px;
      background: white;
}
QPushButton#pushButton_2 {
      border: 1px solid red;
      border-radius: 10px;
}
```

这段代码更改了所有按钮的样式，以及 pushButton_2 按钮的某些属性。我们保持了 pushButton_3 的样式表不变。现在，这些按钮将呈现图 1.4 所示的外观。

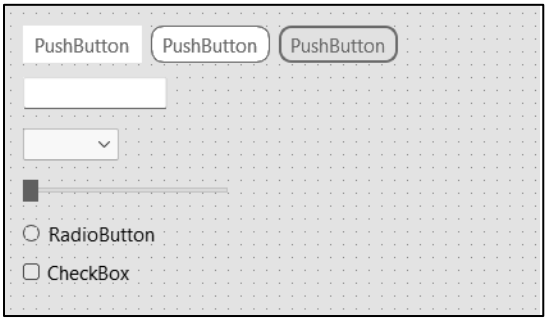

图 1.4　为每个按钮应用不同的样式

（9）第一组样式表将把所有 QPushButton 类型的组件更改为无边框的白色矩形按钮，并配以红色文本。第二组样式表仅更改名为 pushButton_2 的特定 QpushButton 组件的边框。注意，pushButton_2 的背景色和文本色仍然保持为白色和红色，因为我们在第二组样式表中没有覆盖它们，因此它将恢复到第一组样式表中描述的样式，因为该样式适用于所

有 QpushButton 组件。第三个按钮的文本颜色也变为红色，因为我们在第三组样式表中没有描述 Color 属性。

（10）创建另一组使用通用选择器的样式表，并使用以下代码。

```
* {
    background: qradialgradient(cx: 0.3, cy: -0.4, fx: 0.3, fy:
    -0.4, radius: 1.35, stop: 0 #fff, stop: 1 #888);
    color: rgb(255, 255, 255);
    border: 1px solid #ffffff;
}
```

（11）通用选择器将影响所有组件，无论其类型如何。因此，前述样式表将为所有组件的背景应用一种漂亮的渐变色，并将其文本设置为白色，同时设置一个像素的实线轮廓，该轮廓也是白色。我们可以使用 rgb 函数（rgb(255, 255, 255)）或十六进制代码（#ffffff）描述颜色值，而不是直接写出颜色的名称（即白色）。

（12）与之前一样，前述样式表不会影响按钮，因为我们已经为它们指定了自己的样式，这些样式将覆盖通用选择器中描述的一般样式。只需记住，在 Qt 中，当一个组件受到多个样式的影响时，最终将使用更具体的样式。当前，用户界面如图 1.5 所示。

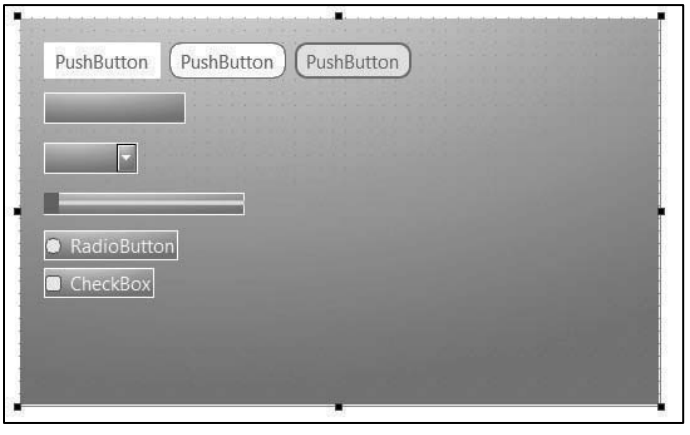

图 1.5　为所有其他组件应用渐变背景

1.3.2　工作方式

如果读者曾参与使用 HTML 和 CSS 进行的网页开发，Qt 的样式表工作方式与 CSS 相同。样式表提供了描述组件呈现的定义——每个组件组中每个元素的颜色是什么，边框值的大小是多少，等等。如果在样式表中指定了组件的名称，它将根据提供的名称改变特定

PushButton 组件的样式。其他组件将不受影响，并将保持默认样式。

要更改组件的名称，可从表单编辑器或对象检查器区域选择组件，并在属性窗口中更改 objectName 属性。如果之前使用了 ID 选择器来更改组件的样式，更改其对象名称将会破坏样式表并丢失样式。要解决这个问题，只需在样式表中也更改对象名称即可。

1.4　使用样式表创建登录界面

本节将学习如何将之前学到的所有知识结合起来，为一个假想的操作系统创建一个模拟的图形登录界面。样式表并不是设计良好用户界面的唯一工具。我们还需要学习如何使用 Qt Designer 中的布局系统整齐地排列组件。

1.4.1　实现方式

具体操作步骤如下所示。

（1）在开始任何操作之前，需要设计图形登录界面的布局。规划对于制作优秀的软件至关重要。以下是一个示例布局设计，用以展示登录界面将如何呈现。图 1.6 所示的简单线条已然足够，只要它能清晰地传达信息。

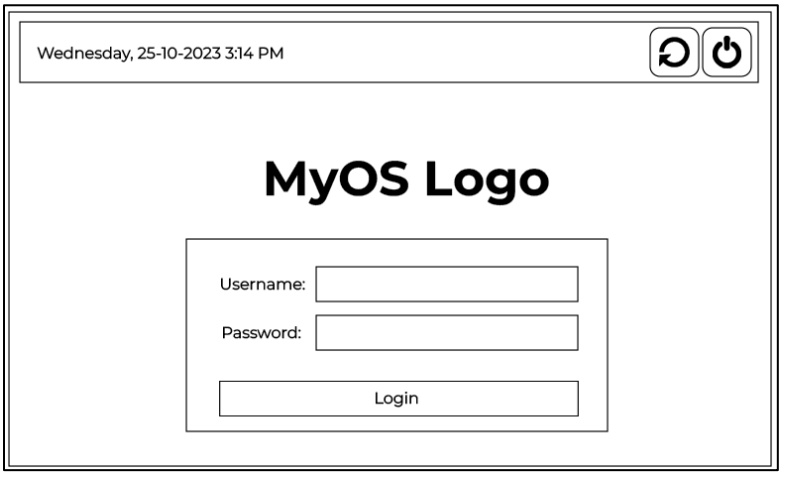

图 1.6　描绘登录界面的简单绘图

（2）再次回到 Qt Designer。

（3）首先在顶部面板放置组件，然后在其下方放置 Logo 和登录表单。

（4）选择主窗口，将其宽度和高度分别从 400 和 300 更改为 800 和 600——我们需要更大的空间来放置所有组件。

（5）从 Widget Box 区域中单击并拖曳 Display Widgets 类别下的标签到表单编辑器。

（6）将标签的 objectName 属性更改为 currentDateTime，并将其文本属性更改为当前的日期和时间以供显示。例如，Wednesday，25-10-2023　3:14 PM。

（7）单击并拖曳 Buttons 类别下的 PushButton 到表单编辑器。重复此过程一次，因为顶部面板上包含两个按钮。将这两个按钮重命名为 restartButton 和 shutdownButton。

（8）选择主窗口，单击表单工具栏上的小图标按钮，当鼠标悬停时显示 Lay Out Vertically。我们将看到组件在主窗口中自动排列，但那还不是我们想要的确切布局。

（9）单击并拖曳 Layouts 类别下的 Horizontal Layout 组件到主窗口。

（10）单击并拖曳两个按钮和文本标签到 Horizontal Layout 中。我们将看到这 3 个组件被水平排列，但垂直方向上，它们位于界面中央。水平排列几乎是正确的，但垂直位置有偏差。

（11）从 Spacers 类别中单击并拖曳一个 Vertical Spacer 组件，将其放置在第（9）步创建的 Horizontal Layout 组件下方（在红色矩形轮廓下方）。所有的组件将被间隔器推到顶部。

（12）在文本标签和两个按钮之间放置一个 Horizontal Spacer 组件，以保持它们之间的距离。这将确保文本标签始终靠左，而按钮则对齐到右侧。

（13）将两个按钮的 Horizontal Policy 和 Vertical Policy 属性都设置为 Fixed，并设置 minimumSize 属性为 55×55。将按钮的 text 属性设置为空，因为我们将使用图标而不是文本。我们将在 1.5 节中学习如何在按钮组件中放置图标。

（14）用户界面应如图 1.7 所示。

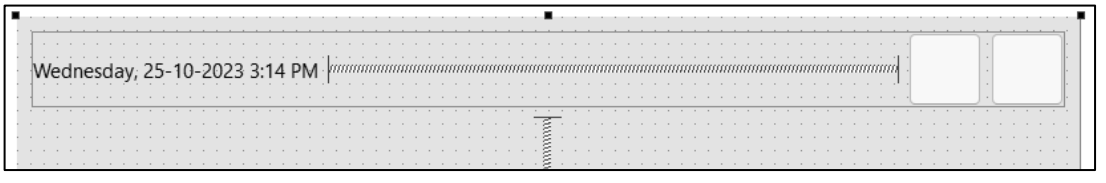

图 1.7　使用水平间隔将文本和按钮分开

接下来将添加 Logo。具体操作步骤如下所示。

（1）在顶部面板和 Vertical Spacer 组件之间添加一个 Horizontal Layout 组件，用作 Logo 的容器。

（2）添加 Horizontal Layout 组件后，我们会发现布局的高度过于细小（几乎为 0），以至于无法向其中添加任何组件。这是因为布局为空，并且被其下方的垂直间隔器压缩至 0 高度。为了解决这个问题，可以将其垂直边距（layoutTopMargin 或 layoutBottomMargin）暂时设置得更大，直到向布局中添加组件。

（3）向刚刚创建的 Horizontal Layout 组件中添加一个 Label 值，并将其重命名为 logo。我们将在 1.5 节中进一步学习如何将图像插入标签中，将其用作 Logo。目前，只需清空其 text 属性，并将 Horizontal Policy 和 Vertical Policy 属性都设置为 Fixed，将最小尺寸属性设置为 150×150。

（4）将布局的垂直边距重新设置为 0。

（5）Logo 现在看起来似乎是不可见的，因此我们将放置一个临时的样式表使其可见，直到我们在 1.5 节中为其添加图像。当前，样式表非常简单，如下所示。

```
border: 1px solid;
```

用户界面如图 1.8 所示。

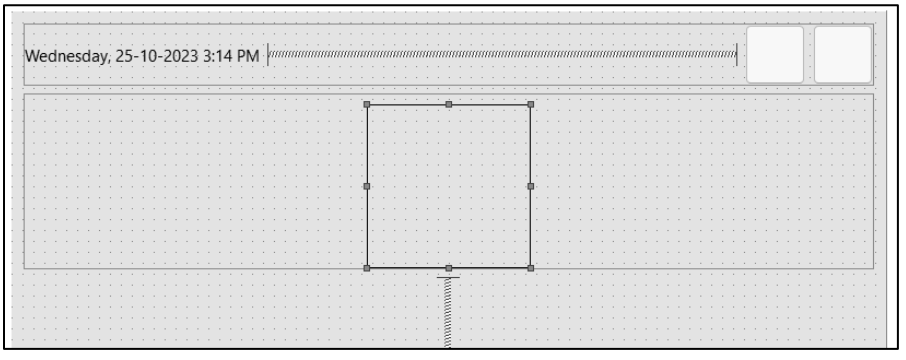

图 1.8　将占位符 Logo 置于中间位置

下面创建登录表单。

（1）在 Logo 布局和 Vertical Spacer 组件之间添加一个 Horizontal Layout 组件。将 layoutTopMargin 属性设置为一个较大的数值（如 100），这样可以更容易地向其中添加组件。

（2）在刚刚创建的 Horizontal Layout 组件内部添加一个 Vertical Layout 组件。该布局将用作登录表单的容器。将其 layoutTopMargin 属性设置为低于 Horizontal Layout 的数值（如 20），以便可以在其中放置组件。

（3）右键单击刚刚创建的 Vertical Layout 组件，选择 Morph into | QWidget。这里，Vertical

Layout 被转换为一个空的组件。这一步是必要的，因为我们将调整登录表单容器的宽度和高度。布局组件不包含任何宽度和高度属性，且只有边距，因为布局会扩展到周围的空间。考虑到它没有任何尺寸属性，这是有道理的。一旦将布局转换为 QWidget 对象，它将自动继承所有来自组件类的属性，这意味着现在可以调整其大小以满足我们的需求。

（4）将刚刚从布局转换而来的 QWidget 对象重命名为 loginForm，并将其 Horizontal Policy 和 Vertical Policy 属性都设置为 Fixed。将最小尺寸参数设置为 350×200。

（5）由于已经将 loginForm 组件放置在 Horizontal Layout 内，我们可以将其 layoutTopMargin 属性重新设置为 0。

（6）为 loginForm 组件添加与 Logo 相同的样式表，以使其暂时可见。然而，这一次，我们需要在前面添加一个 ID 选择器，以便它只将样式应用于 loginForm 而不是其子组件。

```
#loginForm { border: 1px solid; }
```

当前用户界面如图 1.9 所示。

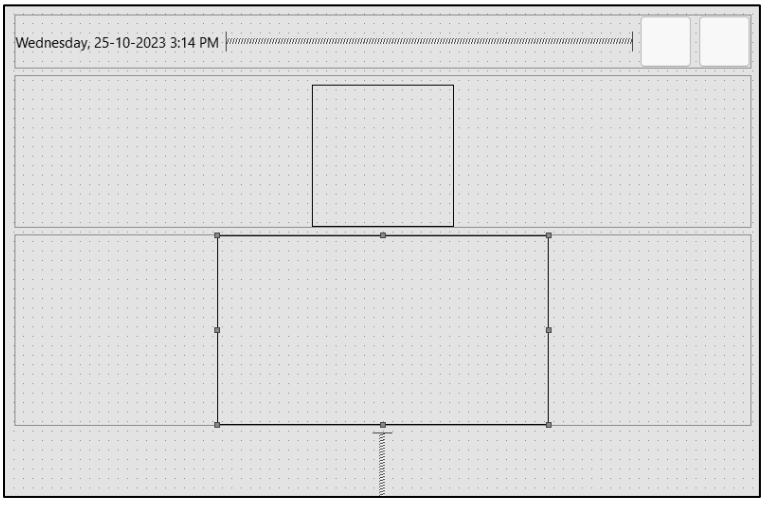

图 1.9　构建登录表单的框架

目前尚未完成登录表单的构建。现在已经为登录表单创建了容器，下面将向表单中添加更多组件。

（1）在登录表单容器中放置两个 Horizontal Layout 组件。我们需要两个布局：一个用于用户名输入框，另一个用于密码输入框。

（2）向刚刚添加的每个布局中添加 Add Label 和 Line Edit 属性。将上方标签的文本属

性更改为 Username:，下方的更改为 Password:。分别将两个行编辑重命名为 username 和 password。

（3）在密码布局下方添加一个按钮，并将其 text 属性更改为 Login。将其重命名为 loginButton。

（4）可以在密码布局和 Login 按钮之间添加一个 Vertical Spacer 组件，以稍微增加它们之间的距离。放置 Vertical Spacer 组件后，将其 sizeType 属性更改为 Fixed，并将 Height 属性更改为 5。

（5）选择 loginForm 容器，并将所有边距设置为 35。这样做是为了在所有边添加一些空间，使登录表单看起来更好。

（6）将 Username、Password 和 loginButton 组件的 Height 属性设置为 25，以避免它们看起来过于拥挤。

当前，用户界面应如图 1.10 所示。

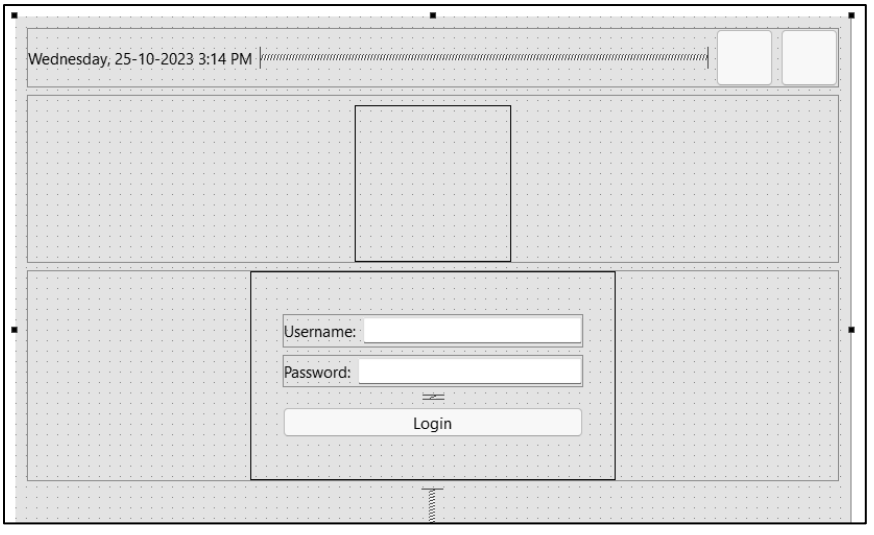

图 1.10　向登录表单添加组件

注意：

或者，也可以使用网格布局来管理 Username 和 Password 输入框，以保持它们的尺寸一致。

可以看到，下方的 Vertical Spacer 组件，Logo 和登录表单都紧贴在主窗口的顶部。Logo 和登录表单应该放置在主窗口的中心位置，而不是顶部。为了解决这个问题，可以按照以

下步骤进行操作。

（1）在顶部面板和 Logo 的布局之间添加另一个 Vertical Spacer 组件。这将抵消底部的间隔器，平衡对齐方式。

（2）如果觉得 Logo 与登录表单过于靠近，可以在 Logo 布局和登录表单的布局之间添加一个 Vertical Spacer 组件。将其 sizeType 属性设置为 Fixed，并将 Height 属性设置为 10。

（3）右键单击顶部面板的布局，选择 Morph into | QWidget。将其重命名为 topPanel。必须将布局转换为 QWidget，因为我们无法对布局应用样式表。这是因为布局除了边距之外没有任何其他属性。

（4）主窗口的边缘周围有一点边距——我们不希望出现这种情况。要去除边距，从对象检查器窗口中选择 centralWidget 对象，该对象位于 MainWindow 面板下，并将所有边距值设置为 0。

（5）单击 Run 按钮（带有绿色箭头图标）运行项目。如果一切顺利，我们应该看到图 1.11 所示的效果。

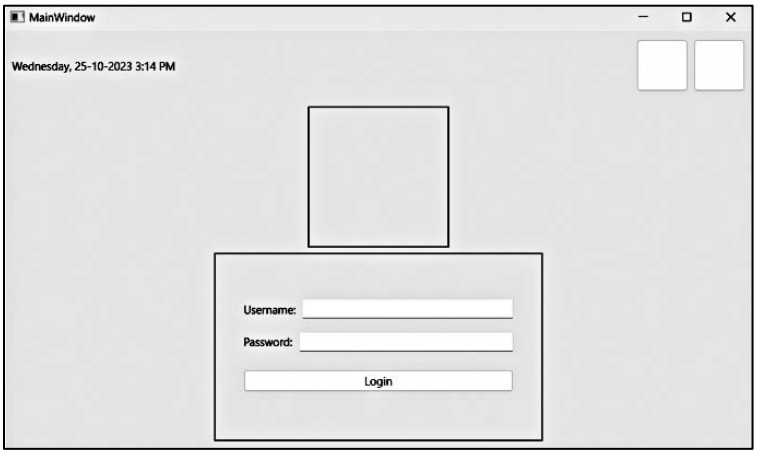

图 1.11　我们已经完成了布局——至少目前是这样

（6）现在，让我们使用样式表来装饰用户界面。由于所有重要的组件都已经被赋予了对象名称，我们可以更容易地从主窗口为它们应用样式表，因为我们只需要将样式表写在主窗口，并让它们沿层次结构树继承下来。

（7）在对象检查器区域右键单击 MainWindow，然后选择 Change styleSheet...。

（8）将以下代码添加到样式表中。

```
#centralWidget { background: rgba(32, 80, 96, 100); }
```

（9）主窗口的背景颜色将会改变。我们将在 1.5 节中学习如何使用图像作为背景。所以，当前颜色只是暂时的。

（10）在 Qt 中，如果想要应用样式到主窗口本身，必须将其应用到其 centralWidget 组件而不是主窗口，因为窗口只是一个容器。

（11）为顶部面板添加一个漂亮的渐变色。

```
#topPanel {
    background-color:  qlineargradient(spread:reflect,  x1:0.5,  y1:0,
x2:0, y2:0, stop:0 rgba(91, 204, 233, 100), stop:1 rgba(32, 80, 96, 100));
}
```

（12）将登录表单应用黑色并使其呈现半透明效果。我们还将通过设置 border-radius 属性，使登录表单容器的边角略微圆滑。

```
#loginForm {
    background: rgba(0, 0, 0, 80);
    border-radius: 8px;
}
```

（13）为通用类型的组件应用样式。

```
QLabel { color: white; }
QLineEdit { border-radius: 3px; }
```

（14）前述样式表将把所有标签的文本颜色更改为白色，这包括组件上的文本，因为从内部看，Qt 在带有文本的组件上使用了相同类型的标签。同时，我们使行编辑组件的边角略微圆滑。

（15）为用户界面上的所有按钮应用样式表。

```
QPushButton {
    color: white;
    background-color: #27a9e3;
    border-width: 0px;
    border-radius: 3px;
}
```

（16）前述样式表将所有按钮的文本颜色更改为白色，然后将背景颜色设置为蓝色，并使其边角略微圆滑。

（17）为了进一步增强效果，我们将使用 hover 关键字，使得当鼠标悬停在按钮上时按钮的颜色发生变化。

```
QPushButton:hover { background-color: #66c011; }
```

（18）前述样式表将在鼠标悬停在按钮上时将按钮的背景颜色更改为绿色。我们将在1.6节中更详细地讨论这一点。

（19）可以进一步调整组件的大小和边距，使它们看起来更加美观。记得移除在第（6）步直接应用到登录表单上的样式表，以去除其边框线。

（20）登录界面应该如图1.12所示。

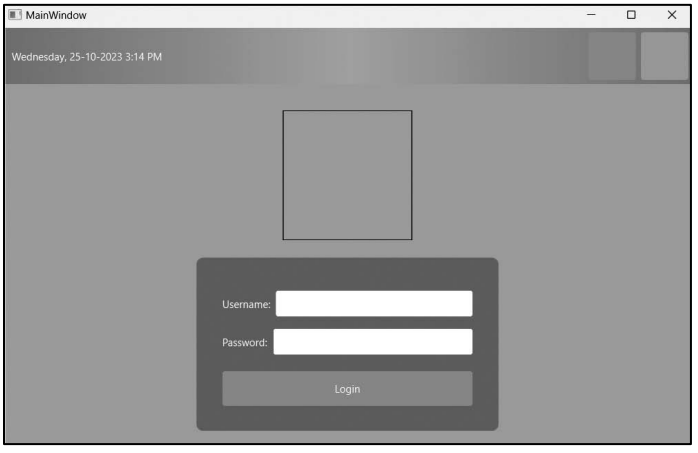

图1.12　为组件应用颜色和样式

1.4.2　工作方式

本例重点介绍了Qt的布局系统。Qt的布局系统允许应用程序GUI通过安排每个组件的子对象自动在给定空间内进行排列。本例中使用的空间填充项有助于将布局中包含的组件向外推动，以沿空间填充项的宽度创建间距。

要将组件定位在布局的中间，必须在布局中放入两个空间填充项：一个在组件的左侧，一个在组件的右侧。然后，这两个空间填充项会将组件推向布局的中间位置。

1.5　在样式表中使用资源

Qt提供了一个平台无关的资源系统，允许将任何类型的文件存储在程序可执行文件中以供后续使用。我们可以在可执行文件中存储的文件类型没有限制——图像、音频、视频、

HTML、XML、文本文件、二进制文件等都是允许的。

资源系统对于将资源文件（如图标和翻译文件）嵌入可执行文件中非常有用，这样应用程序就可以在任何时候访问它们。为了实现这一点，必须在.qrc 文件中告诉 Qt 想要添加到其资源系统中的文件，Qt 将在构建过程中处理其余的事情。

1.5.1　实现方式

要将新的.qrc 文件添加到项目中，可从顶部菜单栏选择 File | New File。然后，在 Files and Classes 类别下选择 Qt，随后选择 Qt Resources File。之后，给它赋予一个名称（如 resources），单击 Next 按钮，然后单击 Finish 按钮。.qrc 文件现在将被创建，并由 Qt Creator 自动打开。我们不必直接以 XML 格式编辑.qrc 文件，因为 Qt Creator 提供了管理资源的用户界面。

要将图像和图标添加到项目中，需要确保图像和图标被放置在项目的目录中。当.qrc 文件在 Qt Creator 中打开时，单击 Add 按钮，然后单击 Add Prefix 按钮。前缀用于对资源进行分类，以便在项目中拥有大量资源时可以更好地管理它们。

（1）将刚刚创建的前缀重命名为/icons。

（2）通过单击 Add 按钮，然后单击 Add Prefix 按钮，创建另一个前缀。

（3）将新前缀重命名为/images。

（4）选择/icon 前缀，然后单击 Add 按钮，接着单击 Add Files 按钮。

（5）将出现一个文件选择窗口，使用该窗口选择所有图标文件。可以通过在键盘上按下 Ctrl 键的同时单击文件来选择多个文件。完成后单击 Open 按钮。

（6）选择/images 前缀，然后单击 Add 按钮，接着单击 Add Files 按钮。文件选择窗口将再次弹出，这一次将选择背景图像。

（7）重复前面的步骤，但这一次将 Logo 图像添加到/images 前缀。完成后不要忘记按下 Ctrl + S 组合键进行保存。.qrc 文件现在应该如图 1.13 所示。

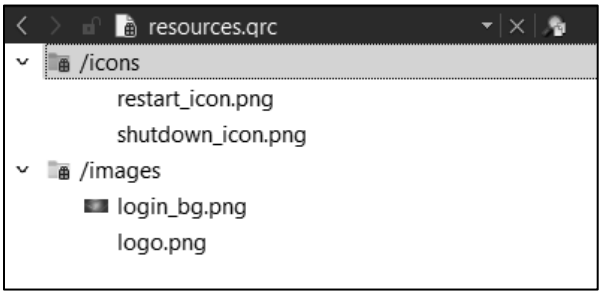

图 1.13　资源文件结构

（8）返回至 mainwindow.ui 文件，并使用我们刚刚添加到项目中的资源。选择位于顶

部面板的重启按钮。在属性编辑器区域向下滚动，直到看到 icon 属性。单击带有下拉箭头图标的小按钮，并从菜单中选择 Choose Resources。

（9）此时将弹出 Choose Resources 窗口。在左侧面板上单击 icons 前缀，并在右侧面板上选择重启图标。随后单击 OK 按钮。

（10）按钮上将出现一个小图标。该图标看起来很小，因为默认图标大小设置为 16×16。将 iconSize 属性更改为 50×50，随后图标将变大。对于关闭按钮，重复前面的步骤，只是这次选择关闭图标。

（11）现在这两个按钮应该如图 1.14 所示。

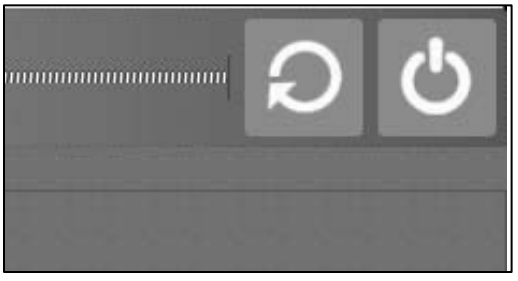

图 1.14　为按钮应用图标

（12）使用添加到资源文件中的图像作为 Logo。选择 Logo 组件，并移除之前添加的用于渲染其轮廓的样式表。

（13）在属性编辑器区域向下滚动，直到看到 pixmap 属性。

（14）单击 pixmap 属性后面的下拉按钮，并从菜单中选择 Choose Resources。选择 Logo 图像并单击 OK 按钮。Logo 的大小不再遵循之前设置的尺寸，它现在遵循图像的实际尺寸。我们不能改变它的尺寸，因为这就是 pixmap 属性的工作方式。

（15）如果想要更多地控制 Logo 的尺寸，可以从 pixmap 属性中移除图像，转而使用样式表。我们可以使用以下代码将图像应用到图标容器。

```
border-image: url(:/images/logo.png);
```

（16）要获取图像的路径，右键单击文件列表窗口中图像的名称，并选择 Copy path。路径将被保存到操作系统剪贴板中。现在，可以直接将其粘贴到前面的样式表中。使用这种方法将确保图像适应应用样式的组件的尺寸。现在，Logo 应如图 1.15 所示。

（17）使用样式表将壁纸图像应用到背景。由于背景尺寸会根据窗口大小变化，我们这里不能使用 pixmap。相反，我们将在样式表中使用 border-image 属性。右键单击主窗口并

选择 Change styleSheet...以打开 Edit Style Sheet 窗口。我们将在 centralWidget 组件的样式表下添加下列新代码。

```
#centralWidget {
    background: rgba(32, 80, 96, 100);
    border-image: url(:/images/login_bg.png);
}
```

（18）当前，登录界面如图 1.16 所示。

图 1.15　Logo 出现在登录表单的顶部

图 1.16　最终结果看起来整洁有序

1.5.2　工作方式

Qt 中的资源系统在编译时将二进制文件（如图像和翻译文件）存储在可执行文件中。它读取项目中的资源集合文件（.qrc），以定位需要存储在可执行文件中的文件，并将它们包含在构建过程中。.qrc 文件如下所示。

```
<!DOCTYPE RCC>
<RCC version="1.0">
    <qresource>
          <file>images/copy.png</file>
          <file>images/cut.png</file>
          <file>images/new.png</file>
          <file>images/open.png</file>
          <file>images/paste.png</file>
          <file>images/save.png</file>
```

```
    </qresource>
</RCC>
```

该文件使用 XML 格式存储资源文件的路径，这些路径是相对于包含它们的目录的。列出的资源文件必须位于与.qrc 文件相同的目录中，或其子目录之一中。

1.6　定制属性和子组件

Qt 的样式表系统能够轻松创建令人惊叹且具有专业外观的用户界面。本例将学习如何为组件设置自定义属性，并使用它们在不同样式之间进行切换。

1.6.1　实现方式

按照以下步骤自定义组件属性和子组件。

（1）创建一个新的 Qt 项目。为此目的，这里已经准备了用户界面。用户界面包含左侧的 3 个按钮，以及右侧的带有 3 页的标签组件，如图 1.17 所示。

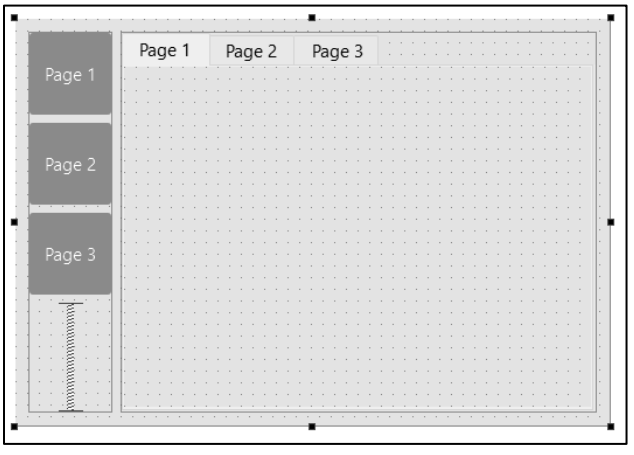

图 1.17　包含 3 个标签页和按钮的基本用户界面

（2）这 3 个按钮是蓝色的，因为已经给主窗口添加了以下样式表（而不是单独的按钮）。

```
QPushButton {
    color: white;
    background-color: #27a9e3;
```

```
    border-width: 0px;
    border-radius: 3px;
}
```

（3）下面将通过向主窗口添加以下样式表以解释 Qt 中的伪状态是什么。读者可能对此已经较为熟悉。

```
QPushButton:hover {
    color: white;
    background-color: #66c011;
    border-width: 0px;
    border-radius: 3px;
}
```

（4）我们在 1.4 节中使用了前述样式表，以在鼠标悬停事件发生时改变按钮的颜色。这是通过 Qt 样式表的伪状态实现的，在这种情况下，伪状态是 hover，它通过冒号与 QPushButton 类分开。每个组件都有一组通用的伪状态，如 active、disabled 和 enabled，以及一组适用于其组件类型的伪状态。例如，open 和 flat 这样的状态可用于 QPushButton，但不适用于 QLineEdit。下面添加 pressed 伪状态，以在用户单击按钮时将其颜色更改为黄色。

```
QPushButton:pressed {
    color: white;
    background-color: yellow;
    border-width: 0px;
    border-radius: 3px;
}
```

（5）伪状态允许用户根据适用于它们的条件加载不同的样式表集合。Qt 通过在 Qt 样式表中实现动态属性，将这一概念推向了更深层次。这使我们能够在满足自定义条件时更改组件的样式表。我们可以利用这一特性，根据在 Qt 中使用自定义属性设置的自定义条件，更改按钮的样式表。首先将把这个样式表添加到主窗口中。

```
QPushButton[pagematches=true] {
    color: white;
    background-color: red;
    border-width: 0px;
    border-radius: 3px;
}
```

（6）如果 pagematches 属性返回 true，则会将按钮的背景颜色更改为红色。该属性在 QPushButton 类中不存在。然而，可以使用 QObject::setProperty()方法将它添加到按钮中。

- 在 mainwindow.cpp 源代码中，在 ui->setupUi(this)之后添加以下代码。

```
ui->button1->setProperty("pagematches", true);
```

上述代码将为第一个按钮添加一个名为 pagematches 的自定义属性，并将它的值设置为 true。这将使第一个按钮默认变为红色。

- 右键单击 Tab Widget，选择 Go to slot….。随后将弹出一个窗口，从列表中选择 currentChanged(int)选项，然后单击 OK 按钮。Qt 将生成一个槽函数（slot function），如下所示。

```
private slots:
void on_tabWidget_currentChanged(int index);
```

- 槽函数将在更改标签组件的页面时被调用。可以通过向槽函数添加代码来决定它应该执行什么操作。对此，打开 mainwindow.cpp 文件并可看到函数的声明。让我们向该函数添加一些代码。

```
void MainWindow::on_tabWidget_currentChanged(int
index) {
    // Set all buttons to false
    ui->button1->setProperty("pagematches", false);
    ui->button2->setProperty("pagematches", false);
    ui->button3->setProperty("pagematches", false);
    // Set one of the buttons to true
    if (0 == index)
        ui->button1->setProperty("pagematches", true);
    else if (index == 1)
        ui->button2->setProperty("pagematches", true);
    else
    ui->button3->setProperty("pagematches", true);
    // Update buttons style
    ui->button1->style()->polish(ui->button1);
    ui->button2->style()->polish(ui->button2);
    ui->button3->style()->polish(ui->button3);
}
```

（7）上述代码在 Tab Widget 切换当前页面时，将所有 3 个按钮的 pagematches 属性设置为 false。在决定哪个按钮应该变为红色之前，确保重置所有设置。

（8）检查事件信号提供的 index 变量；这将告诉我们当前页面的索引号。根据索引号，将其中一个按钮的 pagematches 属性设置为 true。

（9）通过调用 polish()刷新所有 3 个按钮的样式。我们可能还想在 mainwindow.h 中添

加以下头文件。

```
#include <QStyle>
```

（10）构建并运行项目。现在，每当切换 Tab Widget 到不同页面时，应该看到这 3 个按钮变为红色。此外，当鼠标悬停在按钮上时，按钮会变为绿色，单击按钮时则会变黄色，如图 1.18 所示。

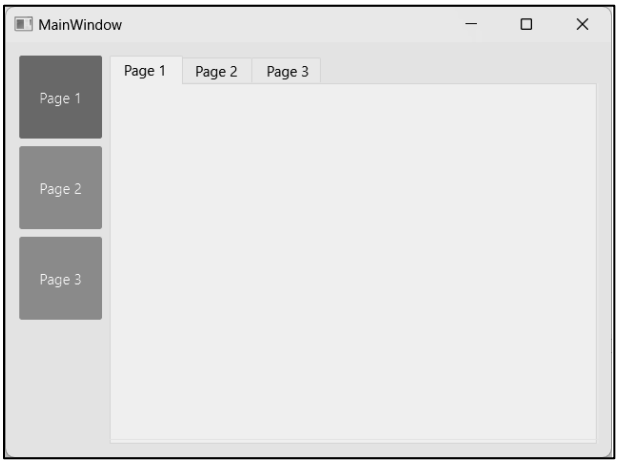

图 1.18　最终的结果

1.6.2　工作方式

Qt 为用户提供了为任何类型的组件添加自定义属性的自由。如果我们希望在满足特定条件时更改某个特定的组件，而 Qt 默认不提供这样的上下文时，自定义属性则非常有用。这允许用户扩展 Qt 的可用性，并使其成为定制解决方案的灵活工具。

例如，如果在主窗口上有一行按钮，并且需要其中一个按钮根据 Tab Widget 当前显示的页面改变其颜色，按钮本身没有办法知道它们何时应该改变颜色，因为 Qt 本身没有内置这种类型情况的上下文。为了解决这个问题，Qt 提供了一种向组件添加自己的属性的方法，它使用了一个名为 QObject::setProperty()的通用函数。要读取自定义属性，可以使用另一个名为 QObject::property()的函数。

接下来将讨论 Qt 样式表中的子组件。通常，一个组件不仅仅是一个单独的对象，而是由多个对象或组件组合而成，形成更复杂的组件。这些对象被称为子组件。

例如，一个微调框组件包含一个输入框、一个向下按钮、一个向上按钮、一个向上箭

头和一个向下箭头，与一些其他组件相比，这相当复杂。在这种情况下，Qt 通过允许使用样式表更改每个子组件，赋予我们更大的灵活性（如果愿意）。我们可以通过在组件类名后面加上双冒号来指定子组件的名称。例如，如果想更改微调框的向下按钮的图像，可以这样编写样式表。

```
QSpinBox::down-button {
    image: url(:/images/spindown.png);
    subcontrol-origin: padding;
    subcontrol-position: right bottom;
}
```

这仅将图像应用到微调框的向下按钮上，而不会应用到组件的其他任何部分。通过结合使用自定义属性、伪状态和子组件，Qt 提供了一种非常灵活的方法来定制用户界面。

注意：

访问以下链接以了解更多关于 Qt 中的伪状态和子组件的信息：http://doc.qt.io/qt-6/stylesheet-reference.html。

1.7　在 Qt 建模语言中进行样式设计

Qt 元编程语言或 Qt 建模语言（QML）是一种受 JavaScript 启发的用户界面标记语言，Qt 用它来设计用户界面。Qt 提供了 Qt Quick 组件（由 QML 技术支持的组件），可以轻松设计适合触摸操作的用户界面，而无须 C++编程。我们将通过本例中提供的步骤，进一步学习如何使用 QML 和 Qt Quick 组件以设计程序的用户界面。

1.7.1　实现方式

按照以下步骤了解 QML 中的样式设计。

（1）从 Qt 6 开始，Qt 公司发布了一个名为 Qt Design Studio 的独立程序，用于开发 Qt Quick 应用程序。它的目的是将设计师和程序员的不同任务分开。因此，如果读者是一名 GUI 设计师，那么应该使用 Qt Design Studio；而如果读者是一名程序员，则应该继续使用 Qt Creator。安装并打开 Qt Design Studio 后，通过单击 Create Project…按钮，或从顶部菜单栏选择 File | New Project…创建一个新项目，如图 1.19 所示。

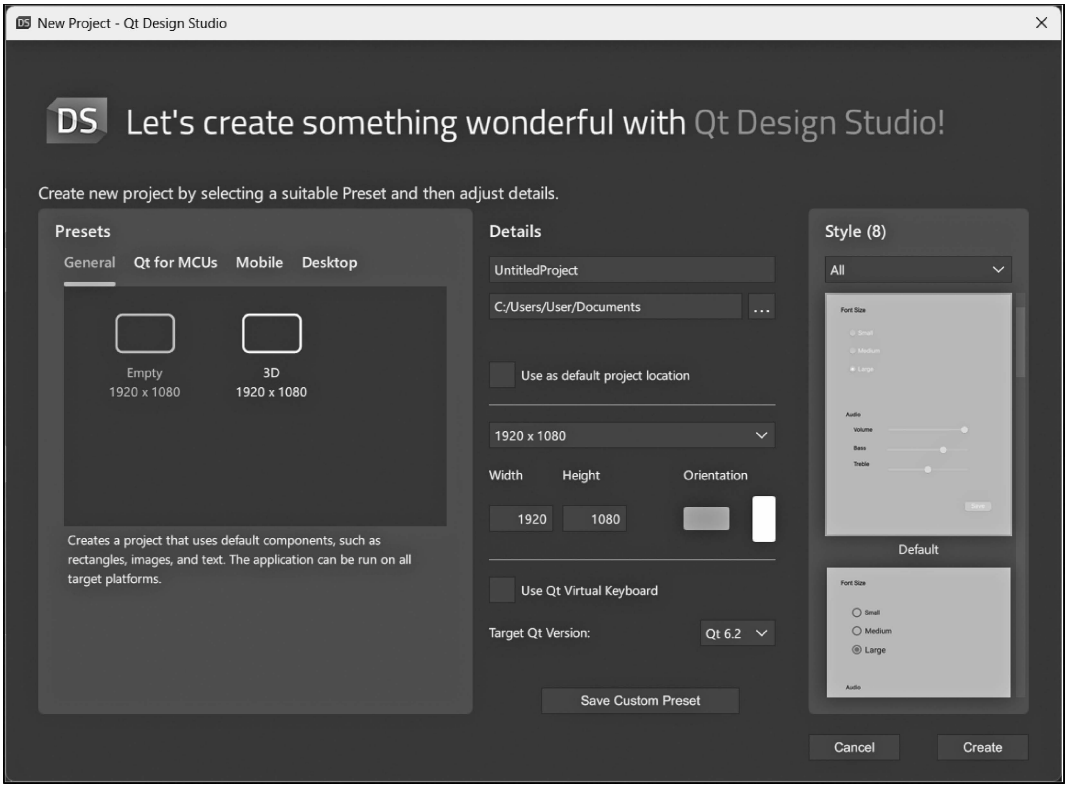

图 1.19　在 Qt Design Studio 中创建一个新的 QML 项目

（2）当 New Project 窗口出现后，输入项目窗口的默认宽度和高度，并为项目输入一个名称。然后，选择希望创建项目的目录，并选择一个默认的 GUI 样式，同时选择一个目标 Qt 版本，并单击 Create 按钮。Qt Quick 项目现在将由 Qt Design Studio 创建。

（3）QML 项目与 C++ Qt 项目之间存在一些差异。我们将看到项目资源中有一个 App.qml 文件。该.qml 文件是使用 QML 标记语言编写的 UI 描述文件。如果双击 main.qml 文件，Qt Creator 将打开脚本编辑器，我们将看到类似这样的内容。

```
import QtQuick 6.2
import QtQuick.Window 6.2
import MyProject
Window {
    width: mainScreen.width
    height: mainScreen.height
```

```
    visible: true
    title: "MyProject"
    Screen01 {
        id: mainScreen
    }
}
```

（4）此文件指示 Qt 创建一个窗口，该窗口加载名为 Screen01 的用户界面，并带有项目名称的窗口标题。Screen01 界面来自另一个名为 Screen01.ui.qml 的文件。

（5）如果打开项目中 scr 文件夹中的 main.cpp 文件，我们将看到以下代码行。

```
QQmlApplicationEngine engine;
const QUrl url(u"qrc:Main/main.qml"_qs);
```

（6）上述代码告诉 Qt 的 QML 引擎在程序启动时加载 main.qml 文件。如果想要加载另一个.qml 文件，我们就知道该去哪里寻找代码了。src 文件夹在 Qt Design Studio 项目中是隐藏的，我们可以在项目目录中找到它。

（7）如果现在构建项目，将得到一个带有简单文本和标有 Press me 的按钮的巨大窗口。当单击按钮时，窗口的背景颜色和文本将会改变，如图 1.20 所示。

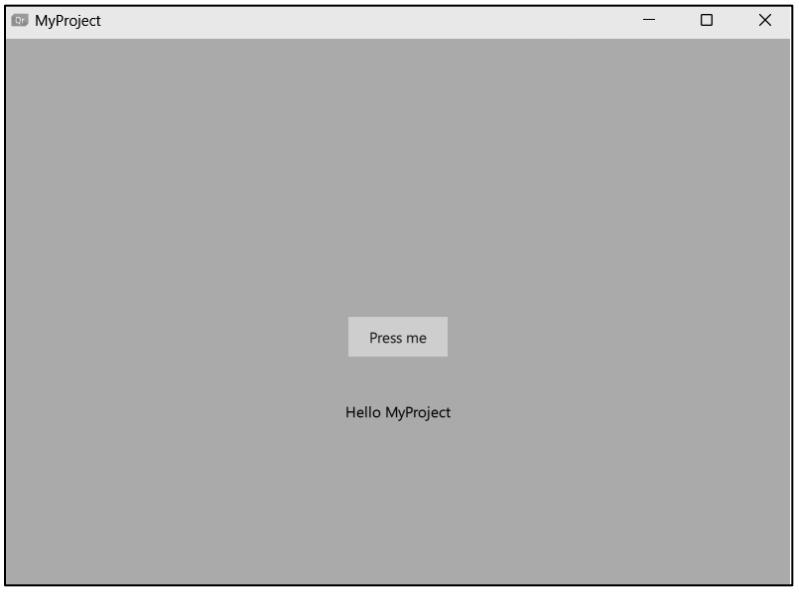

图 1.20　第一个 Qt Quick 程序

（8）要添加用户界面元素，可通过 File | New File…并选择 Files and Classes | Qt Quick

Files 类别下的 Qt Quick UI File 创建一个 Qt Quick 用户界面文件，如图 1.21 所示。

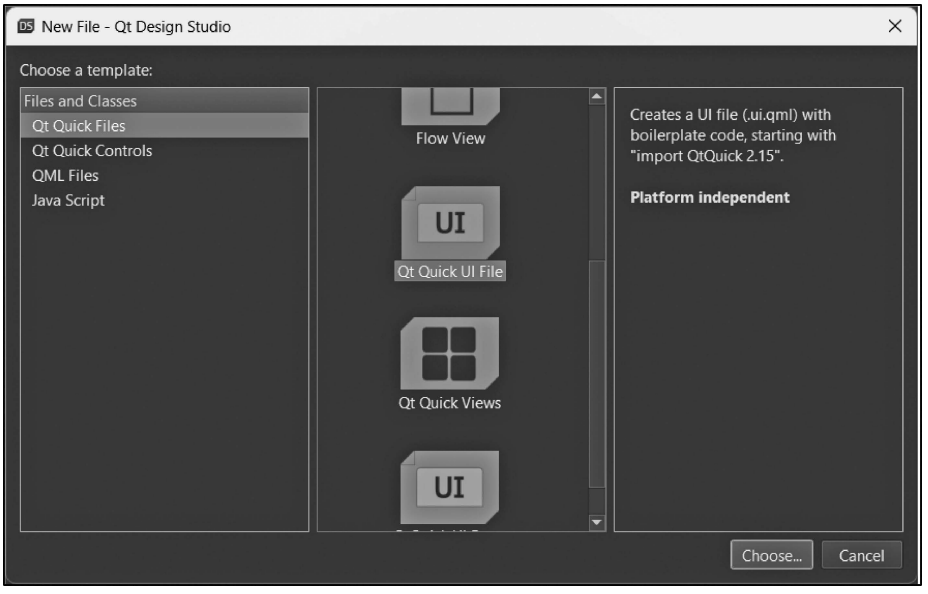

图 1.21　创建新的 Qt Quick UI 文件

（9）将 Component name 设置为 Main，然后单击 Finish 按钮，如图 1.22 所示。

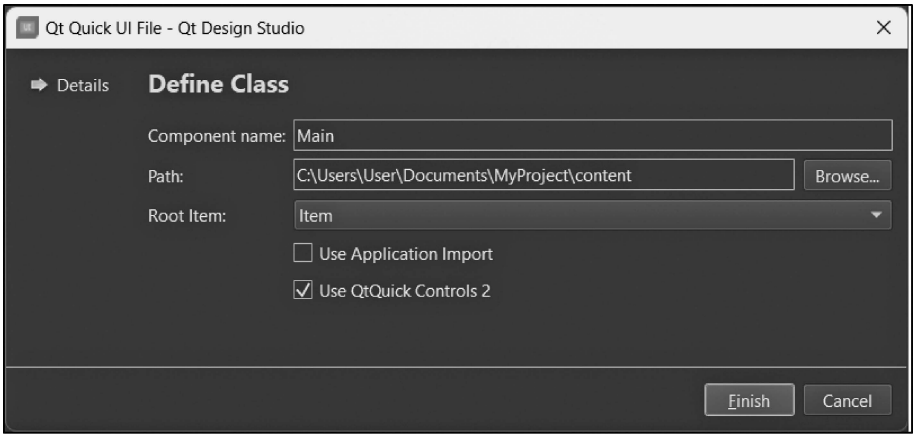

图 1.22　为 Qt Quick 组件赋予一个有意义的名称

（10）一个名为 Main.ui.qml 的新文件已被添加到项目资源中。如果创建时 Qt Design

Studio 没有自动打开它，请尝试双击该文件并打开 Main.ui.qml 文件。我们将看到一个与之前 C++项目中的完全不同的用户界面编辑器。

（11）打开 App.qml 并将 Screen01 替换为 Main，如下所示。

```
Main {
    id: mainScreen
}
```

（12）当 App.qml 由 QML 引擎加载时，它也会将 Main.ui.qml 导入用户界面中，因为 App.qml 文件中现在调用了 Main。Qt 会根据命名约定搜索其.qml 文件，以检查 Main 是否是一个有效的用户界面。这个概念与之前案例中完成的 C++项目类似，App.qml 文件就像 main.cpp 文件，而 Main.ui.qml 就像 MainWindow 类。此外，还可以创建其他用户界面模板并在 App.qml 中使用它们。希望这种比较能让您更容易理解 QML 的工作原理。

（13）打开 Main.ui.qml。我们应该在 Navigator 窗口中只看到一个组件列表：Item。这是窗口的基本布局，且不应被删除。它类似于之前案例中使用的 centralWidget。

（14）目前画布是空的，让我们拖曳一个 Mouse Area 组件和 Text 组件到画布上。调整 Mouse Area 的大小，使其填满整个画布。同时，确保 Mouse Area 和 Text 都放置在 Navigator 面板中的 Item 下，如图 1.23 所示。

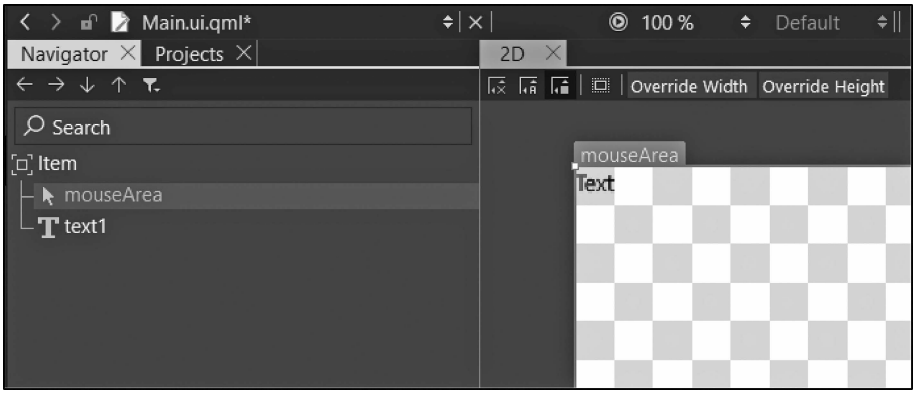

图 1.23　将 Mouse Area 和 Text 拖曳至画布上

（15）Mouse Area 是在鼠标单击或手指触摸（对于移动平台）时被触发的。Mouse Area 也用于 Button 组件中，我们不久将会使用到它。Text 不言而喻：它是一个标签，用于在应用程序中显示一段文本。

（16）在 Navigator 窗口中，可以通过单击组件旁边类似眼睛的图标来隐藏或显示一个组件。当一个组件被隐藏时，它将不会出现在画布或编译后的应用程序中。就像 C++ Qt 项

目中的组件一样，Qt Quick 组件根据父子关系层次排列。所有子组件都将放置在父组件下，且位置缩进。在当前例子中，可以看到 Mouse Area 和 Text 与 Item 相比位置稍微向右，因为它们都是 Item 的子组件。我们可以通过使用 Navigator 窗口中的单击和拖曳方法重新排列父子关系，以及它们在层次结构中的位置。我们可以尝试单击 Text 并将其拖曳至 Mouse Area 上方。随后可以看到 Text 改变了位置，现在位于鼠标区域下方，且缩进更宽，如图 1.24 所示。

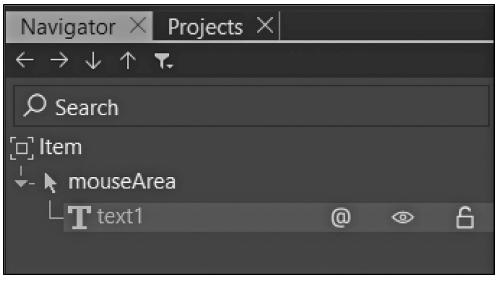

图 1.24　重新排列组件之间的父子关系

（17）可以通过使用 Navigator 窗口顶部的箭头按钮来重新排列它们。对父组件发生的任何事情也会影响其所有子组件，如移动父组件、隐藏和显示父组件。

注意：

可以通过按住鼠标中键（或鼠标滚轮）并移动鼠标平移画布视图。此外，也可以通过按下键盘上的 Ctrl 键并滚动鼠标放大和缩小视图。默认情况下，滚动鼠标会将画布视图上下移动。然而，如果鼠标光标位于画布的水平滚动条上，滚动鼠标将会左右移动视图。

（18）删除 Mouse Area 和 Text，因为我们将学习如何使用 QML 和 Qt Quick 从头开始创建用户界面。

（19）将 Item 组件的大小设置为 800×600，因为需要更大的空间来放置组件。

（20）将之前的 C++项目中使用的图像（即 1.5 节中使用的图像）复制到 QML 项目的文件夹中，因为我们将使用 QML 重新创建相同的登录界面。

（21）将图像添加到资源文件中，以便可以在用户界面中使用它们。

（22）打开 Qt Design Studio 并切换到 Resources 窗口。直接将背景图像拖曳至画布上。在 Properties 窗格上切换到 Layout 标签，然后单击填充锚点按钮，这里用圆圈标示，如图 1.25 所示。这将使背景图像始终贴合窗口大小。

图 1.25　选择填充锚点按钮，使项目跟随其父对象的大小

（23）从 Library 窗口中单击并拖曳一个 Rectangle 组件到画布上，并将其作为程序的顶部面板。

（24）对于顶部面板，启用顶部锚点、左侧锚点和右侧锚点，以便面板紧贴窗口顶部并跟随其宽度。确保所有边距都设置为 0。

（25）转到顶部面板的 Color 属性，并选择 Gradient。将第一种颜色设置为#805bcce9，第二种颜色设置为#80000000。这将创建一个半透明的带有蓝色渐变的面板。

（26）向画布添加一个 Text 组件，并将其设置为顶部面板的子组件。将其 text 属性设置为当前的日期和时间（例如，Wednesday，25-10-2023 3:14 PM）并用于显示。然后，将文本颜色设置为白色。

（27）切换到 Layout 标签，并启用顶部锚点和左侧锚点，以便 Text 组件始终紧贴界面的左上角。

（28）向界面添加一个 Mouse Area 组件，并将其实例大小设置为 50×50。然后，通过在 Navigator 窗口中将其拖曳至顶部面板上方，使其成为顶部面板的子组件。

（29）将 Mouse Area 的颜色设置为蓝色（#27a9e3），并将其圆角设置为 2，使其边角略微圆滑。启用顶部锚点和右侧锚点，使其紧贴窗口的右上角。将顶部锚点的边距设置为 8，右侧锚点的边距设置为 10，以创建一些空间。

（30）打开 Resources 窗口，并将关闭图标拖曳至画布上。使其成为刚刚创建的 Mouse Area 的子组件。然后，启用填充锚点，使其适应鼠标区域的大小。

（31）现在，组件应该在 Navigator 窗口中按图 1.26 中的顺序排列。

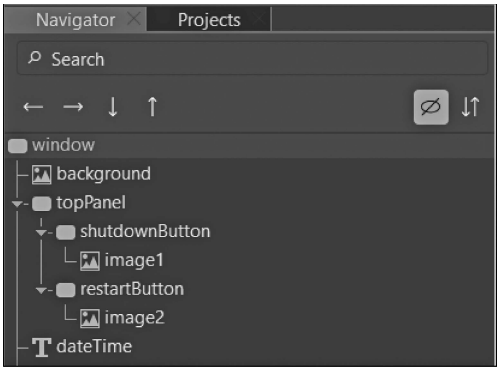

图 1.26　谨慎处理组件之间的父子关系

（32）父子关系和布局锚点对于在主窗口改变大小时保持组件在正确的位置都非常重要。当前，顶部面板应如图 1.27 所示。

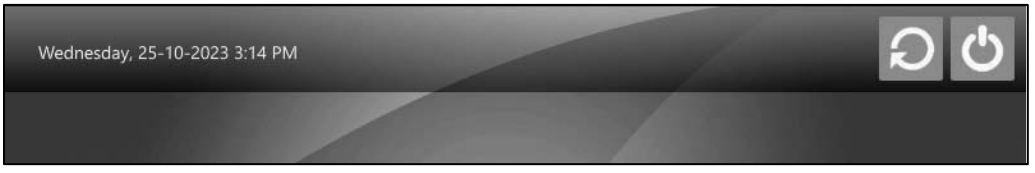

图 1.27　完成顶部面板设计

（33）接下来处理登录表单。通过从 Library 窗口拖曳，向画布添加一个新的 Rectangle 组件。将它调整为 360×200，并将它的圆角设置为 15。

（34）将颜色设置为#80000000，这将把它变为带有 50%透明度的黑色。

（35）启用垂直居中锚点和水平居中锚点，使矩形始终与窗口中心对齐。然后，将垂直居中锚点的边距设置为 100，这样它就会稍微向下移动。这将确保有足够的空间放置 Logo。图 1.28 展示了锚点的设置。

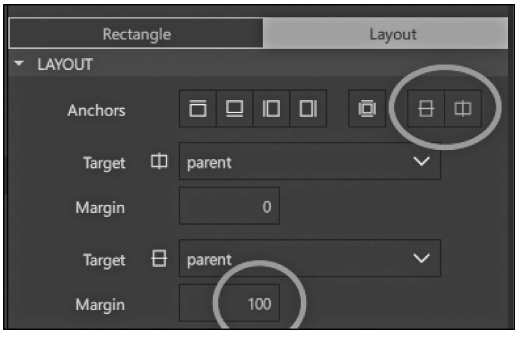

图 1.28　设置对齐和边距

（36）向画布添加 Text 组件。将它们设置为登录表单（Rectangle 组件）的子组件，并将其 text 属性设置为 Username:和 Password:。将其文本颜色更改为白色，并相应地定位它们。这一次不需要设置边距，因为它们将遵循矩形的位置。

（37）向画布添加两个文本输入组件，并将其放置在刚刚创建的 Text 组件旁边。确保文本输入也是登录表单的子组件。由于文本输入不包含任何背景颜色属性，我们需要向画布添加两个矩形来用作它们的背景。

（38）向画布添加两个 Rectangle 组件，并使每个 Rectangle 组件成为刚刚创建的文本输入组件的子组件。将 radius 属性设置为 5，使其具有一些圆角特征。之后，启用填充锚点，使它们能够跟随文本输入组件的大小。

（39）在密码输入框下方创建登录按钮。向画布添加一个 Mouse Area，并使其成为登录表单的子组件。将其调整为我们喜欢的尺寸，并将其移动到适当的位置。

（40）由于 Mouse Area 不包含任何背景颜色属性，我们需要添加一个 Rectangle 组件，并使其成为 Mouse Area 的子组件。将 Rectangle 的颜色设置为蓝色（#27a9e3），并启用填充锚点，使其与 Mouse Area 很好地配合。

（41）向画布添加一个 Text 组件，并使其成为登录按钮的子组件。将其文本颜色更改为白色，并将 text 属性设置为 Login。最后，启用水平居中锚点和垂直居中锚点，使它们与按钮中心对齐。

（42）现在，我们将得到一个与在 C++项目中制作的非常相似的登录表单，如图 1.29 所示。

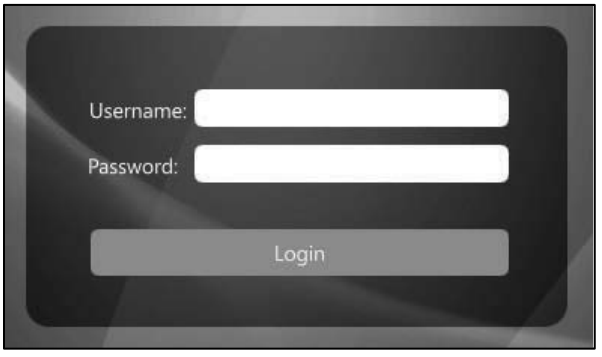

图 1.29　登录表单的最终设计

（43）Logo 的添加过程非常简单。打开 Resources 窗口，并将 Logo 图像拖曳至画布上。

（44）使其成为登录表单的子组件，并将尺寸设置为 512×200。

（45）将其定位在登录表单的顶部。

（46）图 1.30 显示了编译后的用户界面。我们成功地用 QML 和 Qt Quick 重现了 C++ 项目中的登录界面。

图 1.30　最终的结果

1.7.2　工作方式

Qt Quick 编辑器在放置应用程序中的组件方面采用了与表单编辑器截然不同的方法。用户可以决定哪种方法最适合他们的需求。图 1.31 显示了 Qt Quick 设计器的外观。

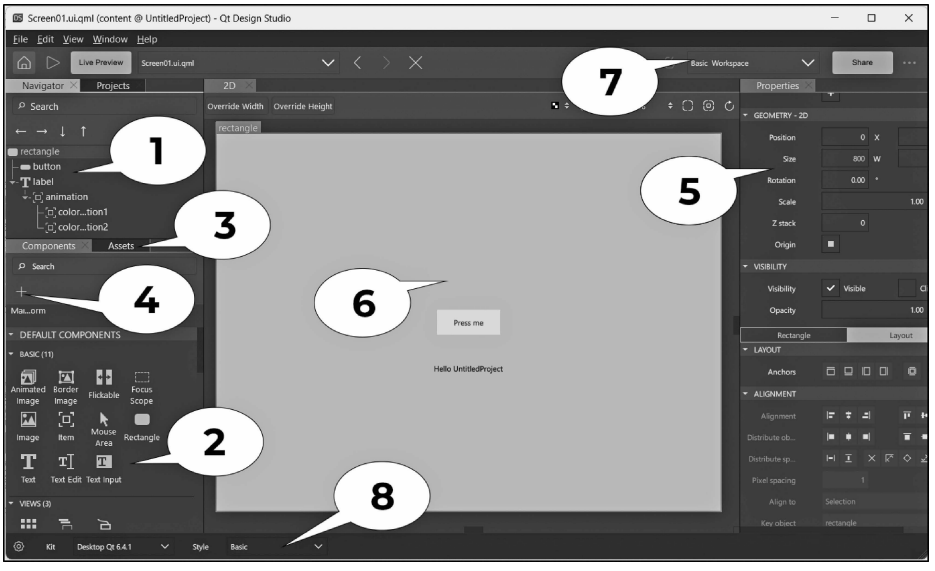

图 1.31　Qt Design Studio 用户界面概览

编辑器用户界面的各个元素如下所示。

（1）Navigator：Navigator 窗口以树状结构显示当前 QML 文件中的项目。它类似于 1.2 节中使用的其他 Qt 设计器中的对象操作窗口。

（2）Library：Library 窗口显示所有在 QML 中可用的 Qt Quick 组件或 Qt Quick 组件。可以单击并将其拖曳至画布窗口以添加到用户界面中。此外，还可以创建自定义 QML 组件并在这里显示。

（3）Assets：Assets 窗口以列表形式显示所有资源，然后可以在 UI 设计中使用。

（4）Add Modules：Add Modules 按钮允许将不同的 QML 模块导入当前的 QML 文件中，如蓝牙模块、WebKit 模块或定位模块，并为 QML 项目添加额外的功能。

（5）Properties：类似于之前使用的属性编辑器区域，QML 设计器中的 Properties 窗格显示所选项的属性。我们还可以在代码编辑器中更改项目属性。

（6）Canvas：Canvas 是创建 QML 组件和设计应用程序的工作区域。

（7）Workspace Selector：Workspace Selector 区域显示了 Qt Design Studio 编辑器中可用的不同布局，允许用户选择适合他们需要的工作区。

（8）Style Selector：Style Selector 区域允许选择不同的样式，以预览应用程序在特定平台上运行时的外观。这对于开发跨平台应用程序非常有用。

1.8　将 QML 对象指针暴露给 C++

有时，我们希望通过 C++脚本修改 QML 对象的属性，如更改标签的文本、隐藏/显示组件或更改其大小。Qt 的 QML 引擎允许将 QML 对象注册到 C++类型，这会自动暴露其所有属性。

1.8.1　实现方式

我们希望在 QML 中创建一个标签并偶尔更改其文本。为了将标签对象暴露给 C++，可以执行以下操作。

（1）在 mylabel.h 中创建一个名为 MyLabel 的 C++类，它继承自 QObject 类。

```
class MyLabel : public QObject {
Q_OBJECT
public:
    // Object pointer
```

```
    QObject* myObject;
    explicit MyLabel(QObject *parent = 0);
    // Must call Q_INVOKABLE so that this function can be used
    // in QML
    Q_INVOKABLE void SetMyObject(QObject* obj);
}
```

（2）在 mylabel.cpp 源文件中，定义一个名为 SetMyObject()的函数以保存对象指针。此函数稍后将在 QML 中的 mylabel.cpp 中被调用。

```
void MyLabel::SetMyObject(QObject* obj) {
    // Set the object pointer
    myObject = obj;
}
```

（3）在 main.cpp 中包含 MyLabel 头文件，并使用 qmlRegisterType()函数将其注册到 QML 引擎。

```
include "mylabel.h"
int main(int argc, char *argv[]) {
    // Register your class to QML
    qmlRegisterType<MyLabel>("MyLabelLib", 1, 0, "MyLabel");
}
```

（4）注意，在 qmlRegisterType()中需要声明 4 个参数。除了声明类名（MyLabel），还需要声明库名（MyLabelLib）及其版本（1.0）。这将用于将类导入 QML 中。

（5）在 QML 中将 QML 引擎映射到标签对象，并在 QML 文件中通过调用 import MyLabelLib 1.0 导入之前在步骤（3）中定义的类库。注意，库名及其版本号必须与在 main.cpp 中声明的内容相匹配；否则，它将抛出错误。在 QML 中声明 MyLabel 并将其 ID 设置为 mylabels 后，调用 mylabel.SetMyObject(myLabel)以在标签初始化后立即将其指针暴露给 C/C++。

```
import MyLabelLib 1.0
ApplicationWindow {
    id: mainWindow
    width: 480
    height: 640
    MyLabel {
            id: mylabel
    }
    Label {
```

```
            id: helloWorldLabel
            text: qsTr("Hello World!")
            Component.onCompleted: {
                  mylabel.SetMyObject(hellowWorldLabel);
            }
    }
}
```

（6）在将标签指针暴露给 C/C++之前，请等待标签完全初始化；否则可能会导致程序崩溃。为确保其完全初始化，请在 Component.onCompleted 内调用 SetMyObject()函数，而不是在任何其他函数或事件回调中调用。现在 QML 标签已经暴露给 C/C++，可以通过调用 setProperty()函数更改其任何属性。例如，可以将其可见性设置为 true，并将文本更改为 Bye bye world!。

```
// Qvariant automatically detects your data type
myObject->setProperty("visible", Qvariant(true));
myObject->setProperty("text", Qvariant("Bye bye world!"));
```

（7）除了更改属性，还可以通过以下代码调用其函数。

```
QVariant returnedValue;
QVariant message = "Hello world!";
QMetaObject::invokeMethod(myObject, "myQMLFunction",
Q_RETURN_ARG(QVariant, returnedValue), Q_ARG(QVariant,
message));
qDebug() << "QML function returned:" <<
returnedValue.toString();
```

（8）简单地说，如果不期望从 invokedMethod()函数中返回任何值，我们可以只调用两个参数的 invokedMethod()函数。

```
QMetaObject::invokeMethod(myObject, "myQMLFunction");
```

1.8.2　工作方式

QML 被设计成可以通过 C++代码进行扩展。Qt QML 模块中的类允许 QML 对象在 C++中被使用和操作，QML 引擎与 Qt 的元对象系统的结合能力允许直接从 QML 调用 C++功能。要向 QML 添加一些 C++数据或应用，它应该来自一个派生自 QObject 的类。QML 对象类型可以从 C++中创建并被监控以访问它们的属性、调用它们的方法和获取它们的信号警报。这是可能的，因为所有 QML 对象类型都是使用派生自 QObject 的类执行

的，允许 QML 引擎通过 Qt 元对象系统强制加载和检查对象。

1.8.3　附加内容

Qt 6 提供了两种不同类型的 GUI 工具包，即 Qt Widgets 和 Qt Quick。它们有各自的优势和长处，赋予程序员设计应用程序界面的能力与自由，而无须担心功能限制和性能问题。

Qt 6 允许选择最适合我们的工作风格和项目需求的最佳方法和编程语言。通过本章的学习，我们将能够迅速使用 Qt 6 创建外观美观且功能齐全的跨平台应用程序。