第 14 章　验证和正则表达式

输入数据通常需要进行过滤或者验证以确保输入是合适的。数字型数据一般需要在一定的范围内。字符串数据通常必须要有一定的格式。输入数据的验证是一项重要的课题，可以按照面向对象的方式使用各 Qt 类的组合来进行处理。本章将讨论一些对输入数据进行验证的有效方法，其中还包括了正则表达式(regular expression)的用法。

验证器(validator)和输入掩码(input mask)让开发人员可以精细地控制 QLineEdit 的行为，可用于对特定的输入类型进行限制。在使用它们之前需要记住的是，Qt 已经为许多常见的值类型提供了一些预定义的输入窗件和输入对话框(QDateEdit，QTimeEdit，QCalendarWidget，QColorDialog，QSpinBox 和 QDoubleSpinBox)，这在让开发人员限制输入值和缩小有效值范围的同时，允许用户轻松选择数值。对于 QDateEdit 来说，开发人员还可以从一系列本地化的日期格式中进行选用。参阅图 9.9，或者运行 Qt 的标准对话框示例程序(Standard Dialogs Example)来感受一下。

14.1　输入掩码

所谓的输入掩码，是一种控制用户在输入窗件中可键入的内容的主动模式。它有助于防止输入某些类型不正确的数据。每个 QLineEdit 都有一个 QString 属性用来存储掩码字符(mask character)——用于那些键入的数据上。输入掩码可以指定在键入 QLineEdit 的字符串中哪个位置处的何种字符是允许的。该字符串由一些特殊的、预定义的掩码字符和一些占据输入字符串相应位置的(可选的)普通字符构成。

表 14.1 中列出的小写版本的掩码字母，详细说明了在该位置上允许但并非必须的相应输入字符。使用 0 而不是 9 表明该位置允许但并非必须有一个 ASCII 数字。#表明该位置允许但并非必须有一个 ASCII 数字或者一个加号(+)或一个减号(-)。此外，表 14.2 中还给出了一些元字符(meta character)。

表 14.1　掩码字符

字　符	相应位置处所需字符	字　符	相应位置处所需字符
A	ASCII 字母型字符——大写或者小写	9	ASCII 数字
N	ASCII 字母数字型字符——大写或者小写	H	十六进制数字
X	任意的 ASCII 字符	B	二进制数字
D	ASCII 非零数字		

表 14.2　掩码元字符

字　符	效　果
>	随后的字母字符是大写的
<	随后的字母字符是小写的
!	结束大小写转换
\	转义字符

为了演示输入掩码的用法，下面给出一个简短的应用程序，它允许用户指定输入掩码字符，然后看看是如何限制输入的。示例 14.1 显示了该类的定义。

示例 14.1　src/validate/inputmask/masktestform.h

```
[ . . . . ]
class MaskTestForm : public QWidget {
   Q_OBJECT
public:
   MaskTestForm();
public slots:
   void showResult();
   void installMask();
   void again();
private:
   QLineEdit* m_InputMask;
   QLineEdit* m_StringEntry;
   QLabel* m_Result;
   void setupForm();
};
[ . . . . ]
```

示例 14.2 中实现了这个类。

示例 14.2　src/validate/inputmask/masktestform.cpp

```
[ . . . . ]
MaskTestForm::MaskTestForm(): m_InputMask(new QLineEdit),
    m_StringEntry(new QLineEdit), m_Result(new QLabel) {
    setupForm();
    move(500, 500); /*Start in mid screen (approx). */
}

void MaskTestForm::setupForm() {
    setWindowTitle("Mask Test Demo");
    QPushButton* againButton = new QPushButton("Another Input Mask", this);
    QPushButton* quitButton = new QPushButton("Quit", this);
    QFormLayout *form = new QFormLayout(this);
    form->addRow("Mask String:", m_InputMask);
    form->addRow("Test Input: ", m_StringEntry);
    form->addRow("Result:", m_Result);
    connect(m_InputMask, SIGNAL(returnPressed()),
            this, SLOT(installMask()));
    connect(m_StringEntry, SIGNAL(returnPressed()),
            this, SLOT(showResult()));
```

```
[ . . . . ]
}
void MaskTestForm::installMask() {
    m_StringEntry->setInputMask(m_InputMask->text());
}
[ . . . . ]
```

图 14.1 给出了应用程序在运行时的屏幕截图

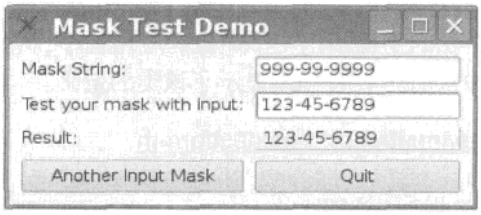

图 14.1 输入掩码

14.1.1 练习：输入掩码

1. 构建和运行 MaskTestForm 应用程序并用它来体验一下下列 inputMask 的值。
   a. AV99e77
   b. \AV99e77
2. 提供一个可接收以下情况的输入掩码(并使用该形式测试你自己的答案)。
   a. 仅允许格式为 123-45-6789 有效社会保障号码
   b. 仅允许格式为 2K3 Y4W(数字和字母相互交替)的加拿大邮政编码
   c. 仅允许格式为 02114-5678 的美国邮政编码
   d. 仅允许格式为(234)345-4567 的美国电话号码
   e. 仅允许格式为 1-123-234-5678 的美国电话号码
3. 能否让最后一个美国电话号码格式开头部分的 1 成为可选项(也就是说，以便也可以接收诸如 123-456-5678 这样的电话号码)？

14.2 验证器

验证器(validator)是可附加到输入窗件(例如 QLineEdit， QSpinBox 和 QComboBox)的不可见对象，能够提供一个检查用户输入的通用框架。Qt 有一个名称为 QValidator 的抽象类，它为所有内置的和自定义的验证器定义了接口。

QValidator 的两个实体子类可以用来对数值范围进行检查：QIntValidator 和 QDoubleValidator。另一个实体子类可以用一个指定的正则表达式来验证字符串。下一节将讨论正则表达式。

QValidator::validate()是一个纯虚函数，它会返回一个枚举值，其含义如下所示。

- Invalid：表达式不满足所要求的条件，进一步输入也于事无补。
- Intermediate：虽然表达式现在不满足所要求的条件，但是进一步的输入可能会产生一个可接受的结果。

- Acceptable：表达式满足所要求的条件。

通过使用 QValidator 类的一些其他成员函数，可以设置 validate()函数使用的条件，例如范围条件)。

一般情况下，正在工作的验证器不允许用户输入可使其返回无效值的数据。

示例 14.3 中给出了一个简短的应用程序，Work-Study Salary Calculator，其中用到了两个数值验证器。该程序可接收用户输入的表示产品信息的一个 int 值和显示其产品信息的一个 double 值并显示它们的乘积。用户按下回车键时，会计算并显示应付总额(Total Pay)的值。

示例 14.3　src/validate/numvalidate/inputform.h

```
[ . . . . ]
class InputForm : public QWidget {
    Q_OBJECT

 public:
    InputForm(int ibot, int itop, double dbot, double dtop);
 public slots:
    void computeResult();
 private:
    void setupForm();
    int m_BotI, m_TopI;
    double m_BotD, m_TopD;
    QLineEdit* m_IntEntry;
    QLineEdit* m_DoubleEntry;
    QLabel* m_Result;
};
[ . . . . ]
```

示例 14.4 中，验证器在构造函数中用范围值进行了初始化，然后又在 setupForm()函数中被指派给相应的输入窗件。

示例 14.4　src/validate/numvalidate/inputform.cpp

```
[ . . . . ]
InputForm::InputForm(int ibot, int itop, double dbot, double dtop):
    m_BotI(ibot), m_TopI(itop), m_BotD(dbot), m_TopD(dtop),
    m_IntEntry(new QLineEdit("0")),
    m_DoubleEntry(new QLineEdit("0")),
    m_Result(new QLabel("0")) {
    setupForm();
    move(500, 500); /*Start in mid screen (approx). */
}

void InputForm::setupForm() {
    [ . . . . ]
    QIntValidator* iValid(new QIntValidator(m_BotI, m_TopI, this));
    QDoubleValidator*
             dValid(new QDoubleValidator(m_BotD, m_TopD, 2, this));
    m_IntEntry->setValidator(iValid);
    m_DoubleEntry->setValidator(dValid);
    connect(m_IntEntry, SIGNAL(returnPressed()),
            this, SLOT(computeResult()));
```

```
    connect(m_DoubleEntry, SIGNAL(returnPressed()),
            this, SLOT(computeResult()));
}
[ . . . . ]
```

程序运行的屏幕截图如图 14.2 所示。

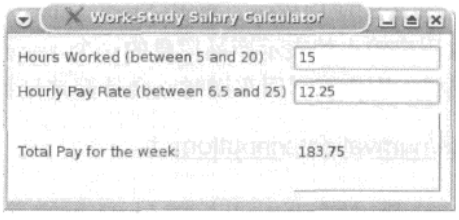

图 14.2 Work Study 计算器

14.2.1 练习：验证器

1. 构建并运行 Work-Study Salary Calculator 应用程序。在第一个 LineEdit 中试着输入一个非 `int` 型值，或者在第二个 LineEdit 中试着输入一个非 `double` 型值。
2. 对于每个 LineEdit，如果试图输入一个超出范围的值会发生什么。例如，对于每小时 3 美元的 2 个小时的工作，其应付总额是多少?

14.3 正则表达式

正则表达式是验证输入、从输入中提取数据以及对输入进行搜索和替换的强大工具。所谓正则表达式，regexp(或者缩写为 regex)，是一种利用模式匹配语言来描述字符串组成限制条件的方式。

正则表达式首先出现在 `vi`，`emacs`，`awk`，`sed` 等工具和 POSIX 标准库中。Perl 是第一个将正则表达式紧密集成到语言中的主流编程语言，那也是人们第一次真正开始学习正则表达式时。人们后来对 Perl 能够识别的正则表达式版本进行了许多改进。这些改进的机制已经成为我们所说的Perl风格的正则表达式扩展的一部分。这些扩展了的正则表达式同样在Java和 Python 中存在。C++0x 和 boost 也提供 C++正则表达式工具。

Qt 提供了一个 `QRegExp` 类，它实现了 Perl 风格的扩展正则表达式语言的大部分功能。

14.3.1 正则表达式语法

与字符串非常相似，正则表达式是一个字符的序列。然而，并非所有字符都只取字面意思。例如，尽管正则表达式中的一个‘a’可以和目标字符串中的‘a’相匹配，但字符‘.’则可以和任意字符相匹配。这里的‘.’就称为元字符(meta-character)。另外一个常用的元字符是‘*’，它可以用来说明目标字符串中可能存在零个或者更多个能够与‘*’前字符串相匹配的字符串。例如，‘a*’将可以和一行中任意数量(包括零个)的‘a’相匹配。如下所示，有许多不同种类的元字符。

下面是一些最经常使用的元字符。

(1) 特殊字符

- .(匹配任何字符)
- \n(匹配换行符)
- \f(匹配换页符)
- \t(匹配制表符)
- \x*hhhh*(匹配一个 Unicode 字符，其对应的码值是范围为 0x0000 到 0xFFFF 之间的一个十六进制数 *hhhh*)

(2) 量词——说明前面的字符(或字符组)在匹配的表达式中可出现次数的修饰符。

- +(出现 1 次或者更多次)
- ?(出现 0 次或者 1 次)
- *(出现 0 次或者更多次)
- {*i*,*j*}(出现至少 *i* 次但不超过 *j* 次)

(3) 字符集——在匹配表达式指定位置允许出现的字符集合。其中还预定义了几个字符集合：

- \s(匹配任何空白符)
- \S(匹配任何非空白符)
- \d(匹配任何数字字符：从‘0’到‘9’)
- \D(匹配任何非数字符合)
- \w(匹配任何“单词”字符，也就是任意的字母、数字或者下画线)
- \W(匹配任意的非单词字符)

字符集也可以使用方括号指定：

- [AEIOU](匹配这五个字符中的任意一个)
- [a-g](短线使此集合可匹配从‘a’到‘g’的字符)
- [^xyz](匹配任何除这三个字符以外的字符)

(4) 分组和捕获字符——(圆括号)是可以用来把字符划分成组的特殊字符。字符组可以是后向引用的。也就是说，如果存在一个匹配，那么分组了的值将可以通过各种方式来捕获和访问。

为简便起见，一般规定在一个正则表达式中最多可以引用 9 个分组，即使用\1 到\9 这样的修饰符。

此外还有一个 QRegExp 成员函数 cap(int nth)，它返回第 *n* 个分组(基于 QString 的形式返回)。

(5) 锚点字符(Anchoring Character)——确定尝试进行匹配操作的边界。

- 脱字符(^)，如果它是正则表达式中的第一个字符，则说明匹配过程从字符串的开头处开始。
- 美元符($)，如果它是正则表达式的最后一个字符，则表明匹配过程直到字符串的结尾才会结束。
- 此外，还有单词边界(\b)断言或者非单词边界(\B)断言，有助于我们关注于正则表达式本身。

表 14.3 中给出了一些正则表达式的例子。

表 14.3 正则表达式的例子

模 式	含 义
hello	匹配字面字符串 hello
c*at	量词：c 出现零次或者多次，at 紧跟其后，例如 at, cat, ccat 等
C?at	匹配 c 出现零次或者一次，之后紧跟 at：仅 at 或者 cat
c.t	c 后面紧跟任意字符，其后面又紧跟 t 的字符串匹配，例如 cat, cot, c3t, c%t 等
c.*t	字符 c 后面紧跟 0 个或者多个任意字符，然后紧跟 t，例如 ct, caaatttt, carsdf$#S8ft 等
ca+t	(量词)'+'意味着前面的字符可以出现一次或者多次，因此符合条件的有 cat, caat, caaaat 等
c\.*t	反斜线在特殊字符之前将会"使其转义"，因此，只有字符串"c.*t"才匹配
c\\\.t	只和字符串"c\.t"匹配
c[0-9a-c]+z	在 c 和 z 之间有一个或者多个字符在集合[0-9a-c]之间，匹配的字符串类似"c312abbaz"和"caa211bac2z"
the (cat|dog) ate (fish|mouse)	(轮流交替)匹配的结果是 the cat ate the fish, the dog ate the mouse, the dog ate the the fish 或者 the cat ate the mouse
\w+	字母数字(单词字符)的序列，与[a-zA-Z0-9]+等价
\W	非单词字符(标点符号、空白符号等)
\s{5}	正好 5 个空白字符(制表符、空白符或者换行符)
^\s+	匹配字符串开头处的一个或者多个空白字符
\s+$	匹配字符串结尾处的一个或者多个空白字符
^Help	如果 Help 出现在字符串的开头，就匹配它
[^Help]	与字符串中任意地方(和元字符^的意思不一样)的除单词 Help 中任一字母之外的任何单个字符相匹配
\S{1,5}	至少 1 个、至多 5 个非空白字符(可打印的字符)
\d	一个数字[0-9](\D 是一个非数字，也就是[^0-9])
\d{3}-\d{4}	7 位电话号码：555-1234
\bm[A-Z]\w+	\b 代表单词边界：mBuffer 匹配而 StreamBuffer 不匹配

注意

反斜线也可用于转义 C++字符串中的特殊字符，因此 C++字符串内部的正则表达式字符串必须"双倍反斜线"。换句话说，每个\都需要变成\\，为了能够正确匹配反斜线字符本身，则需要 4 个反斜线：\\\\。

注意

如果编译器支持 C++0x，或许希望能够在正则表达式中使用原始引用字符串(raw quoted string)，以避免双倍转义反斜线。

```
R"(The String Data \ Stuff " )"
R"delimiter(The String Data \ Stuff " )delimiter"
```

当然，正则表达式还包括很多内容，还是非常值得花费一些时间来彻底地搞明白它。QRegExp 的文档是开始学习它的好地方。如果需要在更大的范围讨论，推荐参考文献[Friedl 98]。

在此期间，也可以继续探索 QRegExp 的功能，利用来自诺基亚 Qt 的一个例子来测试自己的正则表达式。在 src/regex-tester 目录中可以找到其代码。图 14.3 给出了该程序正在运行时的屏幕截图。

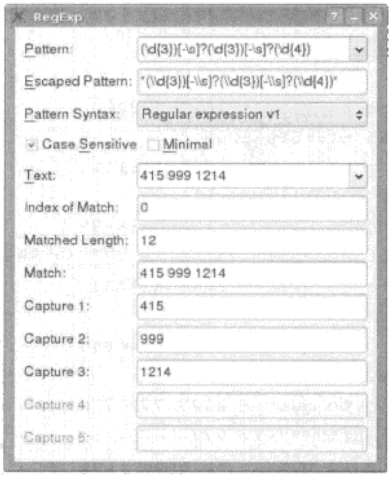

图 14.3　正则表达式测试程序

14.3.2　正则表达式：电话号码识别

14.3.2.1　问题描述

在几乎所有的应用程序中，都有这样一种需求：用一种简单但通用的方式来指定程序运行过程中输入数据时必须满足的一些条件。例如：

- 在美式地址中，每个邮政区号都是一个 5 位数，后接可选的短线(-)和一个 4 位数。
- 美式电话号码是由 10 位数字组成的，通常按照 3 + 3 + 4 的格式分组，此外还可能包含有一些可选的括号、短线以及一个可选的起始值 1。
- 对于美国州名的缩写，必须是 50 个已经得到认可的缩写语之一。

那么应该如何使用面向对象的方式来给定满足上述条件的输入数据的格式?

假设你想要编写这样一个程序，它可识别电话号码的格式并可以接收来自各个国家的各种电话号码，那么需要考虑下面的一些问题：

- 任何美国/加拿大格式的电话号码数字必须形如 *AAA EEE NNNN*，其中 *A* 为地区编码，*E* 为交换中心号码，*N* 为电话号码。
- 对于其他国家电话号码的格式[①]，可以假定其格式必须是 *CC MM*(或 *CCC MM*)后面紧跟 *NN NN NNN* 或者 *NNNNNNN*，此处的 *C* 代表国家，*M* 代表市府代码，*N* 则代表当地的号码数字。
- 在数字簇之间可能会存在短画线或空格。
- 在国家代码之前可能会有+或者 00。

① 在欧洲地区的电话号码情况相当复杂，许多专家已为之付出了多年的努力，希望能够开发一个系统，使其能够被所有欧盟成员国接受。通过访问这里给出的维基百科页面可大致知道其中所要考虑的问题并得到一些灵感。网址为：http://en.wikipedia.org/wiki/ Telephone_numbers_in_Europe 。

现在试想一下，如果仅使用 C++中可用的标准工具，那么应该如何编写这样一个应用程序呢？很有可能需要为每种格式编写一段长长的解析程序。示例 14.5 给出了这样一个程序的预期输出结果。

示例 14.5 src/regexp/testphone.txt

```
src/regexp> ./testphone
Enter a phone number (or q to quit): 16175738000
 validated: (US/Canada) +1 617-573-8000
Enter a phone number (or q to quit): 680111111111
 validated: (Palau) + 680 (0)11-11-11-111
Enter a phone number (or q to quit): 777888888888
 validated: (Unknown - but possibly valid) + 777 (0)88-88-88-888
Enter a phone number (or q to quit): 86333333333
 validated: (China) + 86 (0)33-33-33-333
Enter a phone number (or q to quit): 962444444444
 validated: (Jordan) + 962 (0)44-44-44-444
Enter a phone number (or q to quit): 56777777777
 validated: (Chile) + 56 (0)77-77-77-777
Enter a phone number (or q to quit): 351666666666
 validated: (Portugal) + 351 (0)66-66-66-666
Enter a phone number (or q to quit): 31888888888
 validated: (Netherlands) + 31 (0)88-88-88-888
Enter a phone number (or q to quit): 20398478
Unknown format
Enter a phone number (or q to quit): 2828282828282
Unknown format
Enter a phone number (or q to quit): q
src/regexp>
```

示例 14.6 是一种 C 风格的解决方案，用来说明如何使用 QRegExp 来解决这一问题。

示例 14.6 src/regexp/testphoneread.cpp

```
[ . . . . ]
QRegExp filtercharacters ("[\\s-\\+\\(\\)\\-]");                    1

QRegExp usformat                                                    2
("(\\+?1[- ]?)?\\(?(\\d{3})\\)?[\\s-]?(\\d{3})[\\s-]?(\\d{4})");

QRegExp genformat
("(00)?([[3-9]\\d{1,2})(\\d{2})(\\d{7})$");                         3

QRegExp genformat2
("(\\d\\d)(\\d\\d)(\\d{3})");                                        4


QString countryName(QString ccode) {
   if(ccode == "31") return "Netherlands";
   else if(ccode == "351") return "Portugal";
[ . . . . ]
   //Add more codes as needed ..."
   else return "Unknown - but possibly valid";
}
```

```
QString stdinReadPhone() {                                          5
   QString str;
   bool knownFormat=false;
   do {                                                             6
      cout << "Enter a phone number (or q to quit): ";
      cout.flush();
      str = cin.readLine();
      if (str=="q")
         return str;
      str.remove(filtercharacters);                                 7
      if (genformat.exactMatch(str)) {
         QString country = genformat.cap(2);
         QString citycode = genformat.cap(3);
         QString rest = genformat.cap(4);
         if (genformat2.exactMatch(rest)) {
            knownFormat = true;
            QString number = QString("%1-%2-%3")
                             .arg(genformat2.cap(1))
                             .arg(genformat2.cap(2))
                             .arg(genformat2.cap(3));
            str = QString("(%1) + %2 (0)%3-%4").arg(countryName(country))
                    .arg(country).arg(citycode).arg(number);
         }
      }
[ . . . . ]
      if (not knownFormat) {
         cout << "Unknown format" << endl;
      }
   } while (not knownFormat) ;
   return str;
}

int main() {
    QString str;
    do {
        str =  stdinReadPhone();
        if (str != "q")
            cout << " validated: " << str << endl;
    } while (str != "q");
    return 0;
}
[ . . . . ]
```

1 从用户提供的字符串中移除这些字符。

2 所有美式电话号码都有一个国家代码 1，并且拥有 3 + 3 + 4 = 10 位数字。这些数字之间的空白符、短画线和圆括号都应被忽略掉，但是这些符号能够帮助我们更好地理解电话号码。

3 欧洲国家的电话号码以 3 或者 4 开头，拉丁美洲以 5 开头，东南亚和印度洋地区以 6 开头，东亚地区以 8 开头，而亚洲的中部、南部和西部以 9 开头。国家代码的长度可能是两位数字也可能是三位数字。本地电话号码通常是 2 + 2 + 7 = 11 位或者 3 + 2 + 7 = 12 位。这个程序不会试图解析市府编码。

4 最后 7 位数字会被排列成 2 + 2 + 3 的形式。
5 确保用户输入的电话号码字符串必须遵守一个正则表达式并可从其中提取出合适的内容。返回一个适当格式的电话号码。
6 不停询问，直到得到一个有效的号码。
7 移除所有的短画线、空白符号、括号等。

在一个这样的程序中，用户的所有响应都会在他输入字符并按下回车键之后由 QRegExp 进行检查。没有办法可以阻止用户在输入流中键入那些不合适的字符。

14.3.3 练习：正则表达式

1. 许多操作系统都会保存一个单词列表，而各种程序都会使用此列表中的单词来进行拼写检查。在类*nix 系统中，这个单词列表通常(至少是间接地)命名为“words”。在类*nix 系统中，可以通过输入如下命令来确定该文件的位置：

```
locate words | grep dict
```

用 grep 过滤此命令的输出结果，可以将输出结果缩减为仅包含"dict"的那些行。在确定了操作系统中的单词列表文件之后，试着编写一个程序，从此文件中读入文本行，然后采用合适的正则表达式(或者，如果不太可能这样做，试试其他方法)来显示所有符合下列条件的单词。

- a 以一对重复字母开头的单词
- b 以“gory‘结尾的单词
- c 有重复字母超过一个的单词
- d 是回文的单词(即顺读和倒读都相同的单词)
- e 由按照字母表顺序严格升序排列的字母组成的单词(例如，knot)

如果在操作系统中无法找到合适的单词列表，可以使用本书源代码安装包中提供的文件：src/downloads/canadian-english-small[①]。如果之前没有这方面的准备，就需要用命令 gunzip canadian-english-small.gz 将其解压缩出来。

2. 编写一个程序，使其使用正则表达式来从 HTML 文件中提取超链接。超链接的形式如下：

```
<a href="http://www.web.www/location/page.html">The Label</a>
```

在输入文件中遇到每个超链接时，只打印其 URL 和标签，两者之间用 Tab 键分隔。需要记住的是，可选的空白字符可以在之前的示例模式中的不同部位出现。用一系列含有超链接的不同 Web 页面来测试该程序，并确认程序的确可以捕捉到所有的链接。

3. 假定你刚刚换了一家公司，并且要打算复用那些为前公司编写的开源代码。现在，需要将源代码中的所有数据成员都进行重命名。之前的公司对于数据成员的命名方式类似于：mVarName。然而，新公司希望它们的具有这样的命名方式：m_varName。请用 QDirIterator 对找到的每个文件中的文字都执行替换，以便让所有的数据成员都可以与新公司的编码标准一致。

① 可从附录中的 dist 目录中下载该文件。

14.4　正则表达式验证

QRegExpValidator 类使用了 QRegExp 来验证输入字符串。示例 14.7 中给出了一个包含 QRegExpValidator 和一些输入窗件的主窗口。

示例 14.7　src/validate/regexval/rinputform.h

```
[ . . . . ]

class RinputForm : public QWidget {
    Q_OBJECT
 public:
    explicit RinputForm(QWidget* parent=0);
    void setupForm();
 public slots:
    void computeResult();
 private:
    QLineEdit* m_PhoneEntry;
    QLabel* m_PhoneResult;
    QString m_Phone;
    static QRegExp s_PhoneFormat;
};
[ . . . . ]
```

我们借用了示例 14.6 中的正则表达式，用它来初始化示例 14.8 中的静态 QRegExp。示例 14.8 接受用户输入的一个电话号码，且只在通过有效性验证之后才能将其显示出来。值得注意的是，14.1.1 节中提出问题 3 时我们知道，美国式电话号码中起始的 1 是可有可无的。

示例 14.8　src/validate/regexval/rinputform.cpp

```
[ . . . . ]
QRegExp RinputForm::s_PhoneFormat(
  "(\\+?1[- ]?)?\\(?(\\d{3,3})\\)?[\\s-]?(\\d{3,3})[\\s-]?(\\d{4,4})");

RinputForm::RinputForm(QWidget* parent)
:   QWidget(parent),
    m_PhoneEntry(new QLineEdit),
    m_PhoneResult(new QLabel) {
    setupForm();
    move(500, 500); /*Start in mid screen (approx). */
}

void RinputForm::setupForm() {
    [ . . . . ]
    QRegExpValidator*
          phoneValid(new QRegExpValidator(s_PhoneFormat, this));
    m_PhoneEntry->setValidator(phoneValid);
    connect(m_PhoneEntry, SIGNAL(returnPressed()),
            this, SLOT(computeResult()));
}

void RinputForm::computeResult() {
    m_Phone = m_PhoneEntry->text();
```

```
    if (s_PhoneFormat.exactMatch(m_Phone)) {
        QString areacode = s_PhoneFormat.cap(2);
        QString exchange = s_PhoneFormat.cap(3);
        QString number = s_PhoneFormat.cap(4);
        m_PhoneResult->setText(QString("(US/Canada) +1 %1-%2-%3")
              .arg(areacode).arg(exchange).arg(number));
    }
}
[ . . . . ]
```

QRegExpValidator 类不会接收任何可能产生无效结果的字符。图 14.4 给出了该程序运行过程中的一个屏幕截图。

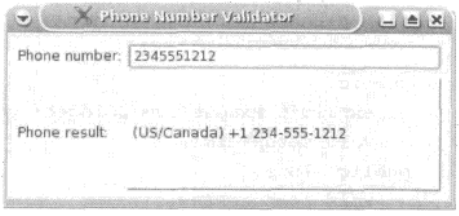

图 14.4 电话号码验证器

QValidator 为输入数据验证提供了一种强大的机制。Qt 提供了两个数值范围验证器和一个正则表达式验证器。如果使用借助数字范围和正则表达式尚无法完成所需验证，可以针对具体情况较为容易地扩展并生成 QValidator 的自定义子类。

14.5 子类化 QValidator

当验证用户输入的需求超出了简单的数字范围或者正则表达式验证验证的能力时，可以通过对 QValidator 派生来定义自己的验证器类。在接下来的例子中，我们定义回文(palindrome)为这样一种字符串，在忽略大小写、空格和标点符号时，它从前往后或者从后往前读起来都相同。遗憾的是，还没有哪一个简单的正则表达式可以用来判定任意长度的字符串是否是一个回文字符串。示例 14.9 给出了一个 Palindate，这是一个可用来验证所给定的字符串是否是一个回文字符串的 QValidator。

示例 14.9 src/validate/palindrome/palindate.h

```
#include <QValidator>
#include <QString>

class Palindate : public QValidator {
   Q_OBJECT
public:
   explicit Palindate(QObject* parent = 0);
   QValidator::State validate(QString& input, int&) const;
};
```

QValidator 类有一个成员函数必须重载：validate()，其定义如示例 14.10 所示。该函数生成给定字符串的小写副本 inpStr，并从中移除所有的空白字符和标点符号。然后，它会把 inpStr 和 revStr 字符串作比较，revStr 字符串中含有同样的字符，但顺序与之相反。

示例 14.10 src/validate/palindrome/palindate.cpp

```
[ . . . . ]
QValidator::State Palindate::validate(QString& str, int& ) const {
   QString inpStr(str.toLower());
   QString skipchars("-_!,;. \t");
```

```
    foreach(QChar ch, skipchars)
        inpStr = inpStr.remove(ch);                                    1
    QString revStr;                                                    2
    for(int i=inpStr.length(); i > 0; --i)
        revStr.append(inpStr[i-1]);
    if(inpStr == revStr)
        return Acceptable;
    else
        return Intermediate;
}
```

1 用 regex 来做的话可能会更快一些。

2 令人诧异的是，没有 `reverse()` 函数。

示例 14.11 定义了一个包含 `QLineEdit` 的窗件以用于测试该验证器。

示例 14.11　src/validate/palindrome/palindromeform.h

```
[ . . . . ]
class PalindromeForm : public QWidget {
    Q_OBJECT
public:
    PalindromeForm(QWidget* parent=0);
    QString getPalindrome();
public slots:
    void showResult();
    void again();
private:
    Palindate* m_Palindate;
    QLineEdit* m_LineEdit;
    QLabel* m_Result;
    QString m_InputString;
    void setupForm();
};
[ . . . . ]
```

示例 14.12 中构建了一些窗件，并在 `QLineEdit` 上设置了一个 Palindate 验证器。

示例 14.12　src/validate/palindrome/palindromeform.cpp

```
[ . . . . ]
PalindromeForm::PalindromeForm(QWidget* parent) : QWidget(parent),
  m_Palindate(new Palindate),
  m_LineEdit(new QLineEdit),
  m_Result(new QLabel) {
    setupForm();
}

void PalindromeForm::setupForm() {
    setWindowTitle("Palindrome Checker");
    m_LineEdit->setValidator(m_Palindate);
    connect(m_LineEdit, SIGNAL(returnPressed()),
            this, SLOT(showResult()));
[ . . . . ]
}
```

```
void PalindromeForm::showResult() {
   QString str = m_LineEdit->text();
   int pos(0);
   if(m_Palindate->validate(str,pos) == QValidator::Acceptable) {
      m_InputString = str;
      m_Result->setText("Valid Palindrome!");
   }
   else {
      m_InputString = "";
      m_Result->setText("Not a Palindrome!");
   }
}
[ . . . . ]
```

图 14.5 中给出了程序运行过程中的一个屏幕截图。

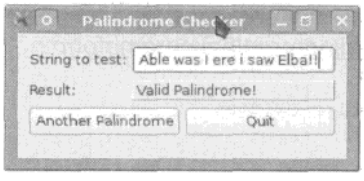

图 14.5 回文检验器

14.6 练习：验证和正则表达式

图 14.6 Address 窗体

1. 设计一个如图 14.6 所示的 Address 窗体。
   当用户从组合框中选择一个国家时，程序必须在三个 QLineEdit 上设置一个适当的验证器，即 stateEdit，zipEdit 和 phoneEdit。同样，对话框的标签应当以下列方式发生变化。
   a. 美国
      - 将 zipLabel 设置成 Zip。
      - 将 stateLabel 设置成 State。
   b. 加拿大
      - 将 zipLabel 设置成 Postal Code。
      - 将 stateLabel 设置成 Province。
   c. 用适当的验证器在选项中另外添加一个国家。
   确保永不接受非法输入(同时还要禁用 OK 按钮)。单击 OK 按钮后，应关闭对话框并打印出每个域的值。如果愿意，可以从 handout/validation/addressform.ui 文件开始。
2. 国际标准书号(International Standard Book Number，ISBN)是一种用于书籍和其他出版物的编码系统。ISBN 的大小可以不同(10 位和 13 位)，它们可以分解成标识语种、出版社和书名的子字符串。此外，最右端的那位数字是一个校验数字，它可从其他位中计算出来以用于错误校验[①]。

① Wikipedia 中有一篇解释该系统的有趣文章，参见 http://en.wikipedia.org/wiki/ISBN。

假定 ISBN-10 编码(包括校验位)看起来是这个样子：

```
d1d2d3...d9d10
```

那么，下式必须得平衡：

```
(10*d1 + 9*d2 + 8*d3 + ... + 2*d9 + d10) % 11 = 0
```

如果所需的校验位是 10,可以在那个位置上放一个字母 X。定义一个当且仅当 ISBN-10 编码的校验位正确时方可接受的 `QValidator`。可以用 0-306-40615-2(有效)和 0-306-40615-5(无效)对其进行测试。

3. 找出(例如，从维基百科的文章中[①])如何验证 ISBN-13 编码校验位的方法，并定义一个当且仅当 ISBN-13 编码的校验位正确时方可接受的 `QValidator`。
4. 你能只用一个 `QValidator` 就既可处理 ISBN-10 编码又可处理 ISBN-13 编码吗？假设仅需验证校验位。

14.7　复习题

1. 什么是输入掩码？输入掩码可以用来干什么？
2. 什么是正则表达式？可以使用正则表达式来做些什么工作？
3. 什么是验证器？它有什么用处？
4. 说明三种不同类型的验证器。
5. 什么是正则表达式的元字符？有四种元字符：量词、字符集、分组和锚点。针对每种元字符给出一个具体的例子并解释其具体含义。
6. 为什么需要扩展 `QValidator`？

① 参见http://en.wikipedia.org/wiki/ISBN。