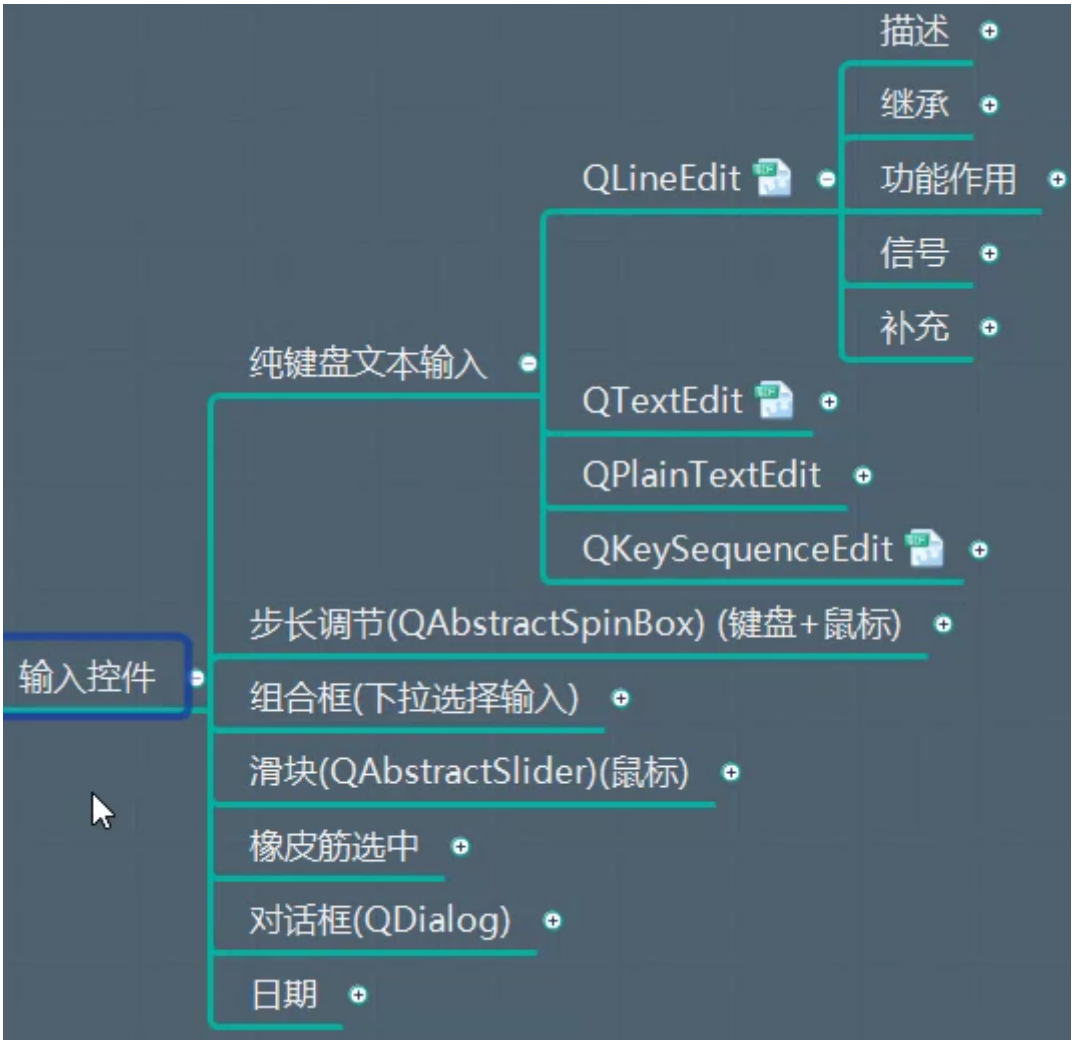
[笔记][LIKE-Python-GUI编程-PyQt5][16]

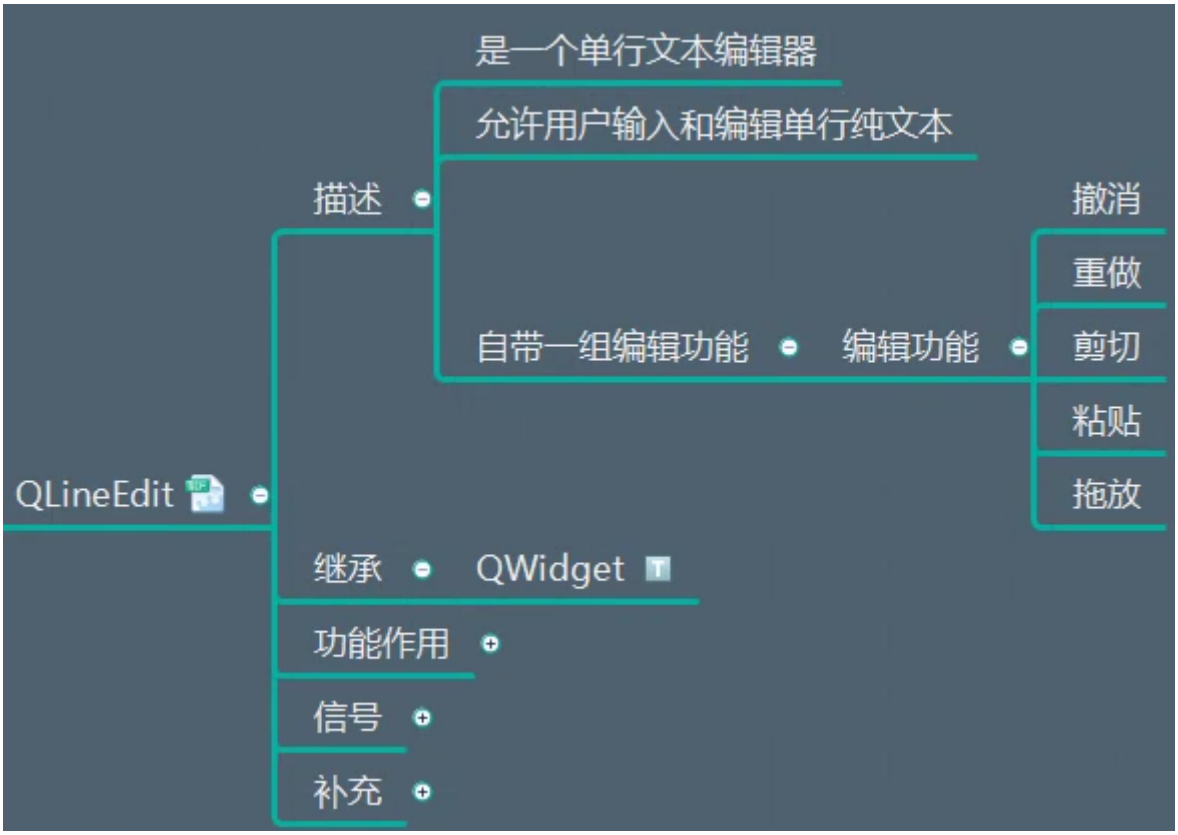
PyQt5

---

112. Python-GUI编程-PyQt5-QLineEdit-简介

输入控件的功能是采集信息。

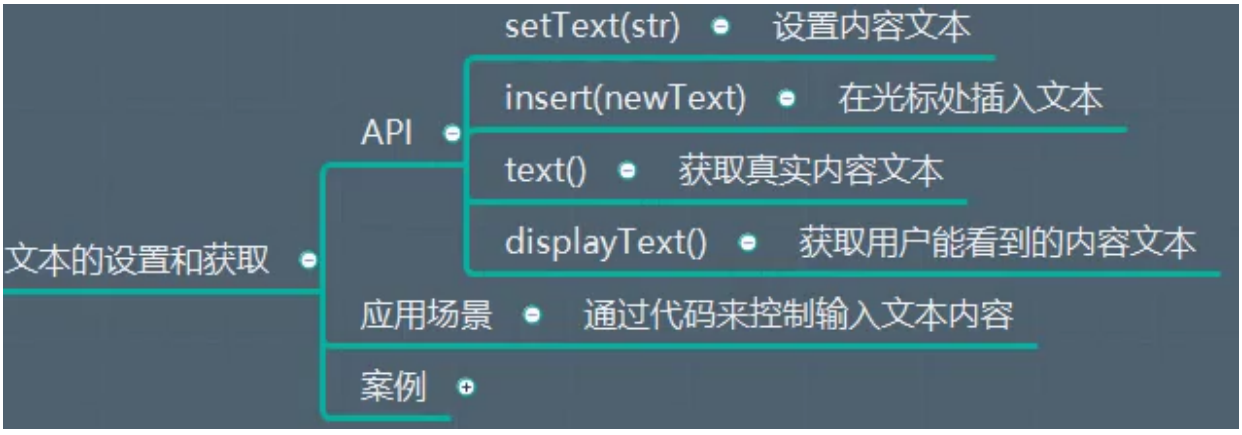

输入控件
纯键盘文本输入
步长调节(QAbstractSpinBox) (键盘+鼠标)
组合框(下拉选择输入)
滑块(QAbstractSlider)(鼠标)
橡皮筋选中
对话框(QDialog)
日期
QLineEdit
描述
继承
功能作用
信号
补充
QTextEdit
QPlainTextEdit
QKeySequenceEdit


QLineEdit
描述
是一个单行文本编辑器
允许用户输入和编辑单行纯文本
自带一组编辑功能
编辑功能
撤消
重做
剪切
粘贴
拖放
继承
QWidget
功能作用
信号
补充

113. Python-GUI编程-PyQt5-QLineEdit-控件创建

```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('QLineEdit控件创建')
window.resize(500, 500)

# QLineEdit 继承自 QWidget
# le = QLineEdit(window)
le = QLineEdit('单行文本编辑器', window)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

114. Python-GUI编程-PyQt5-QLineEdit-文本的设置和获取-API



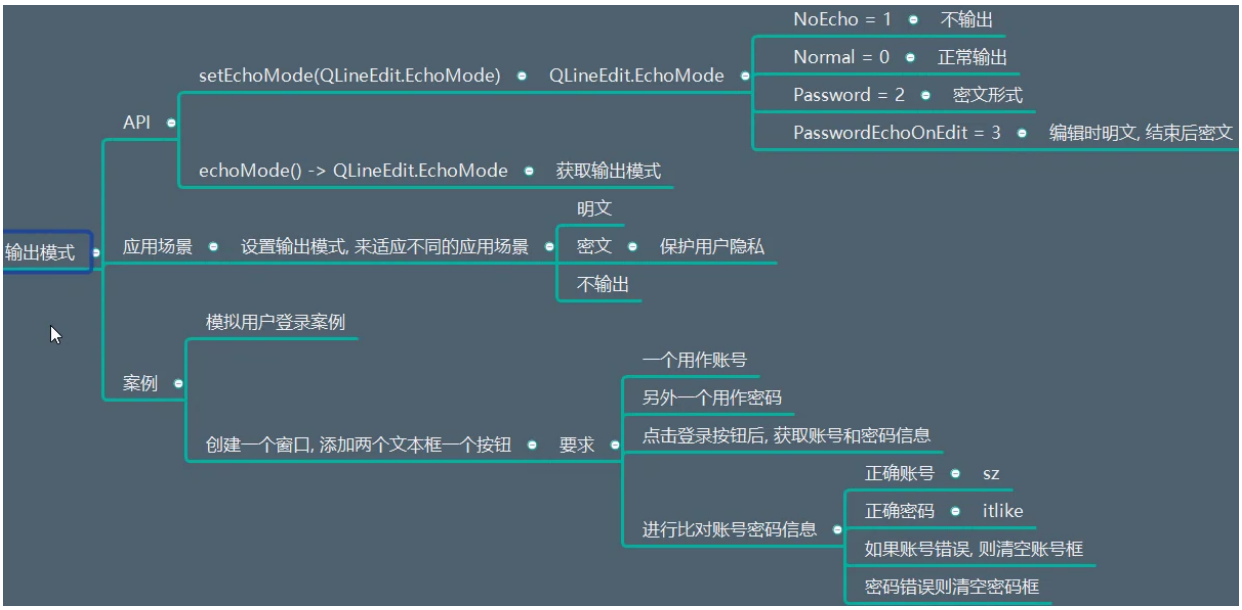

```python
# 0. 导入需要的包和模块
```

```python
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('文本设置和获取')
window.resize(500, 500)

# 其实这个构造函数就是帮我们调用了一个 setText() 方法
le = QLineEdit('单行文本编辑器', window)
# 文本设置
le.setText('进击的团子')
# 在光标处插入
# le.insert('666')

# 在光标处插入文本
btn = QPushButton(window)
btn.setText('插入')
btn.move(100, 100)
btn.clicked.connect(lambda: le.insert('666'))

# 获取文本
btn2 = QPushButton(window)
btn2.setText('获取')
btn2.move(100, 200)
btn2.clicked.connect(lambda: print('文本内容为：', le.text()))

# 获取显示的文本 displayText()

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

115. Python-GUI编程-PyQt5-QLineEdit-文本的设置和获取-案例

```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('案例')
window.resize(500, 500)

le_a = QLineEdit(window)
le_a.move(100, 100)

le_b = QLineEdit(window)
le_b.move(100, 200)

copy_btn = QPushButton(window)
# 注意这里的 setText 是 QAbstractButton 的方法
# 而 le_b.setText 是 QLineEdit 的方法
copy_btn.setText('复制')
copy_btn.move(100, 300)


def copy_text():
    # 获取文本框a的内容
    text = le_a.text()
    # 把上面的内容，设置到文本框b
    le_b.setText(text)

    # 如果是插入文本，就有问题了，因为它不会清除之前的文本！
    # 所以要先清空内容
    # le_b.setText('')
    # le_b.insert(text)


copy_btn.clicked.connect(copy_text)

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

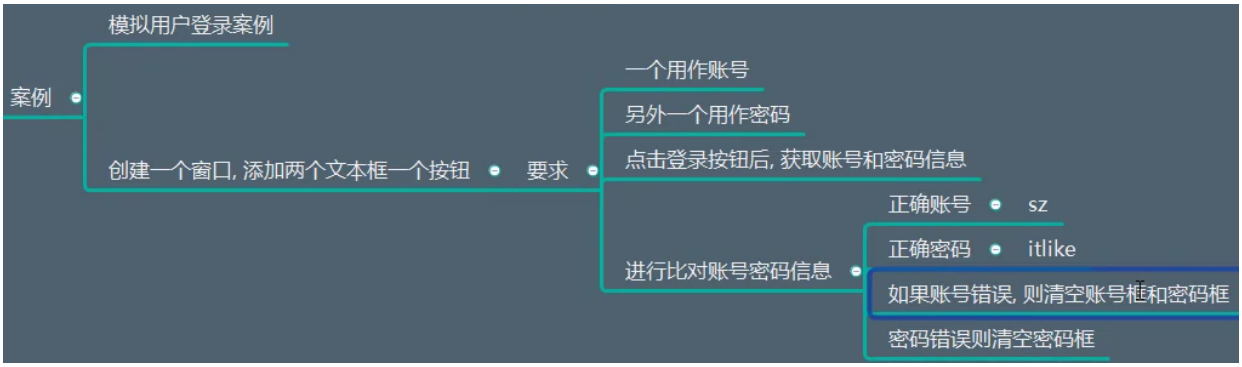
116. Python-GUI编程-PyQt5-QLineEdit-文本输出模式-API




```python
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
# 2.1 创建控件
window = QWidget()
# 2.2 设置控件
window.setWindowTitle('输出模式')
window.resize(500, 500)

le = QLineEdit(window)
le.move(100, 100)

# 查看代码文档
# setEchoMode(self, QLineEdit.EchoMode)
# QLineEdit.EchoMode 说明枚举值被定义在类的内部
# 相当于是定义了类属性，通过 类名.类属性名 来访问
# 在类定义中往下翻就能找到
# Normal = 0
# NoEcho = 1
# Password = 2
# PasswordEchoOnEdit = 3

# 不回显：对密码加密的更狠，连位数都不显示
le.setEchoMode(QLineEdit.NoEcho)
```

```
# 普通模式
le.setEchoMode(QLineEdit.Normal)

# 密文模式
le.setEchoMode(QLineEdit.Password)

# 失去焦点变成密文
le.setEchoMode(QLineEdit.PasswordEchoOnEdit)

btn = QPushButton('打印', window)
btn.move(100, 200)
btn.clicked.connect(lambda: print('内容为：', le.text()))
btn.clicked.connect(lambda: print('展示内容为：', le.displayText()))

# 获取当前输出模式
print('当前输出模式为：', le.echoMode())

# 2.3 展示控件
window.show()

# 3. 应用程序的执行，进入到消息循环
sys.exit(app.exec_())
```

117. Python-GUI编程-PyQt5-QLineEdit-文本输出模式-登录案例-上




```
from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
```

```
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        # 添加三个控件
        # 下面有三个灰色波浪线
        # 提示：Instance attribute ... defined outside __init__
        # 假如在调用 setup_ui 之前就调用了 resizeEvent，就会出现问题
        # 因为此时 resizeEvent 里面会访问还没有设置的相关属性
        # 所以要把相关属性的设置放到 __init__ 里面，才会保证第一时间就被设置
        # 但我们为了方便、美观而提取了一个 setup_ui 方法
        # 实际上 setup_ui 这个方法还是在 __init__ 里面的，所以并不会有什么问题
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.login_btn = QPushButton(self)
        self.login_btn.setText('登  录')

    def resizeEvent(self, evt):
        widget_w = 150
        widget_h = 40
        margin = 60
        self.account_le.resize(widget_w, widget_h)
        self.pwd_le.resize(widget_w, widget_h)
        self.login_btn.resize(widget_w, widget_h)

        x = (self.width() - widget_w) / 2

        self.account_le.move(x, self.height() / 5)
        self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
        self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

118. Python-GUI编程-PyQt5-QLineEdit-文本输出模式-登录案例-下

```
# 拓展1：假如账号和密码其中之一没有内容，登录按钮不可用
# 拓展2：账号下面和密码下面都有提示标签，一开始是隐藏的，根据输入展示提示信息
```

```python
# 拓展3：账号和密码是业务逻辑，需要与界面分离

from PyQt5.Qt import *


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        # 添加三个控件
        # 下面有三个灰色波浪线
        # 提示：Instance attribute ... defined outside __init__
        # 假如在调用 setup_ui 之前就调用了 resizeEvent，就会出现问题
        # 因为此时 resizeEvent 里面会访问还没有设置的相关属性
        # 所以要把相关属性的设置放到 __init__ 里面，才会保证第一时间就被设置
        # 但我们为了方便、美观而提取了一个 setup_ui 方法
        # 实际上 setup_ui 这个方法还是在 __init__ 里面的，所以并不会有什么问题
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.pwd_le.setEchoMode(QLineEdit.Password)
        self.login_btn = QPushButton(self)
        self.login_btn.setText('登  录')

        # 连接槽函数
        self.login_btn.clicked.connect(self.login)

    def login(self):
        # 获取账号和密码信息
        account = self.account_le.text()
        pwd = self.pwd_le.text()
        print(f'账号：{account}\n密码：{pwd}')
        if account == 'jpch89':
            if pwd == '666':
                print('登录成功')
            else:
                print('密码错误')
                self.pwd_le.setText('')
                # 密码框自动获取焦点
                # 之所以会失去焦点，是因为点击了登录按钮！
                self.pwd_le.setFocus()
        else:
            print('账号错误')
            self.account_le.setText('')
            self.pwd_le.setText('')
            # 账号文本框自动获取焦点
            self.account_le.setFocus()
```

```
        # 另外一种写法（不用写很多else）：
        # 账号错误!
        # if account != 'jpch89':
            # 进行操作
            # 账号错误，直接返回
            # return None
        # 账号正确，密码错误!
        # if pwd != '666':
            # 进行操作
            # 密码错误，直接返回
            # return None
        # 走到这里，账号密码都正确
        # print('登录成功! ')

    def resizeEvent(self, evt):
        widget_w = 150
        widget_h = 40
        margin = 60
        self.account_le.resize(widget_w, widget_h)
        self.pwd_le.resize(widget_w, widget_h)
        self.login_btn.resize(widget_w, widget_h)

        x = (self.width() - widget_w) / 2

        self.account_le.move(x, self.height() / 5)
        self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
        self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

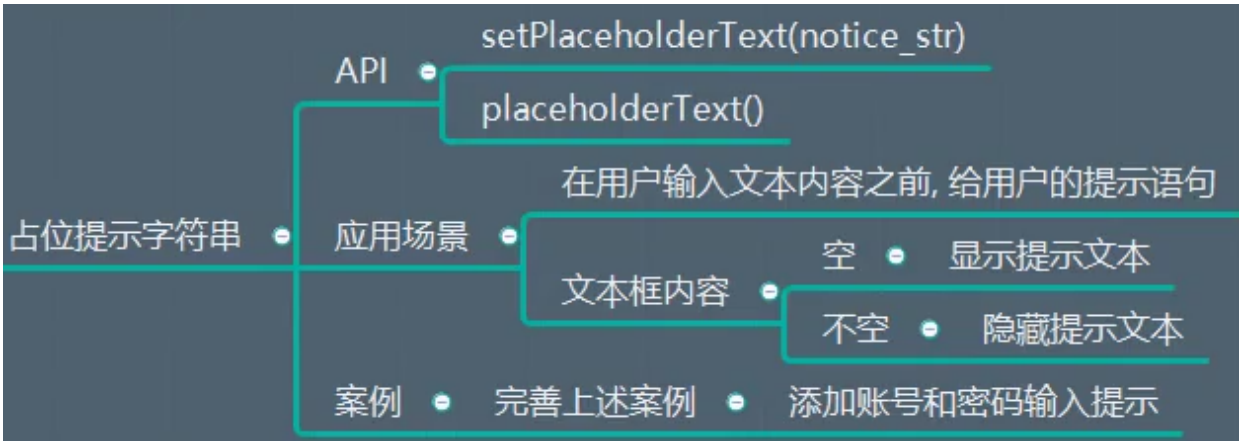
界面与逻辑分离

```
# 拓展1：假如账号和密码其中之一没有内容，登录按钮不可用
# 拓展2：账号下面和密码下面都有提示标签，一开始是隐藏的，根据输入展示提示信息
# 拓展3：账号和密码是业务逻辑，需要与界面分离

from PyQt5.Qt import *
```

```python
class AccountTool:  # 默认继承 object
    ACCOUNT_ERROR = 1
    PWD_ERROR = 2
    SUCCESS = 3

    @staticmethod
    def check_login(account, pwd):
        # 把账号和密码发送给服务器，等待服务器返回结果
        if account != 'jpch89':
            # 返回一个状态
            return AccountTool.ACCOUNT_ERROR
            # 或者通过类名访问也可以
            # return AccountTool.ACCOUNT_ERROR
        if pwd != '666':
            return AccountTool.PWD_ERROR
        return AccountTool.SUCCESS


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.pwd_le.setEchoMode(QLineEdit.Password)
        self.login_btn = QPushButton(self)
        self.login_btn.setText('登  录')

        # 连接槽函数
        self.login_btn.clicked.connect(self.login)

    def login(self):
        # 获取账号和密码信息
        account = self.account_le.text()
        pwd = self.pwd_le.text()
        print(f'账号：{account}\n密码：{pwd}')
        state = AccountTool.check_login(account, pwd)
        if state == AccountTool.ACCOUNT_ERROR:
            print('账号错误')
            self.account_le.setText('')
            self.pwd_le.setText('')
            self.account_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None
```

```
            if state == AccountTool.PWD_ERROR:
                print('密码错误')
                self.pwd_le.setText('')
                self.pwd_le.setFocus()
                # 可写可不写
                # 但是写了可以跳过后续判定
                # 加快程序执行
                return None

            # 这里可以不用判定!
            if state == AccountTool.SUCCESS:
                print('登录成功')

        def resizeEvent(self, evt):
            widget_w = 150
            widget_h = 40
            margin = 60
            self.account_le.resize(widget_w, widget_h)
            self.pwd_le.resize(widget_w, widget_h)
            self.login_btn.resize(widget_w, widget_h)

            x = (self.width() - widget_w) / 2

            self.account_le.move(x, self.height() / 5)
            self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
            self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)


    if __name__ == '__main__':
        import sys

        app = QApplication(sys.argv)

        window = Window()
        window.show()
        sys.exit(app.exec_())
```

119. Python-GUI编程-PyQt5-QLineEdit-占位文本设置




```python
# 拓展1：假如账号和密码其中之一没有内容，登录按钮不可用
# 拓展2：账号下面和密码下面都有提示标签，一开始是隐藏的，根据输入展示提示信息
# 拓展3：账号和密码是业务逻辑，需要与界面分离

from PyQt5.Qt import *


class AccountTool:  # 默认继承 object
    ACCOUNT_ERROR = 1
    PWD_ERROR = 2
    SUCCESS = 3

    @staticmethod
    def check_login(account, pwd):
        # 把账号和密码发送给服务器，等待服务器返回结果
        if account != 'jpch89':
            # 返回一个状态
            return AccountTool.ACCOUNT_ERROR
            # 或者通过类名访问也可以
            # return AccountTool.ACCOUNT_ERROR
        if pwd != '666':
            return AccountTool.PWD_ERROR
        return AccountTool.SUCCESS


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.pwd_le.setEchoMode(QLineEdit.Password)
        self.login_btn = QPushButton(self)
```

```
        self.login_btn.setText('登  录')

        # 占位文本的提示
        self.account_le.setPlaceholderText('请输入账号')
        self.pwd_le.setPlaceholderText('请输入密码')

        # 连接槽函数
        self.login_btn.clicked.connect(self.login)

    def login(self):
        # 获取账号和密码信息
        account = self.account_le.text()
        pwd = self.pwd_le.text()
        print(f'账号：{account}\n密码：{pwd}')
        state = AccountTool.check_login(account, pwd)
        if state == AccountTool.ACCOUNT_ERROR:
            print('账号错误')
            self.account_le.setText('')
            self.pwd_le.setText('')
            self.account_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        if state == AccountTool.PWD_ERROR:
            print('密码错误')
            self.pwd_le.setText('')
            self.pwd_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        # 这里可以不用判定！
        if state == AccountTool.SUCCESS:
            print('登录成功')

    def resizeEvent(self, evt):
        widget_w = 150
        widget_h = 40
        margin = 60
        self.account_le.resize(widget_w, widget_h)
        self.pwd_le.resize(widget_w, widget_h)
        self.login_btn.resize(widget_w, widget_h)

        x = (self.width() - widget_w) / 2

        self.account_le.move(x, self.height() / 5)
        self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
        self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)
```

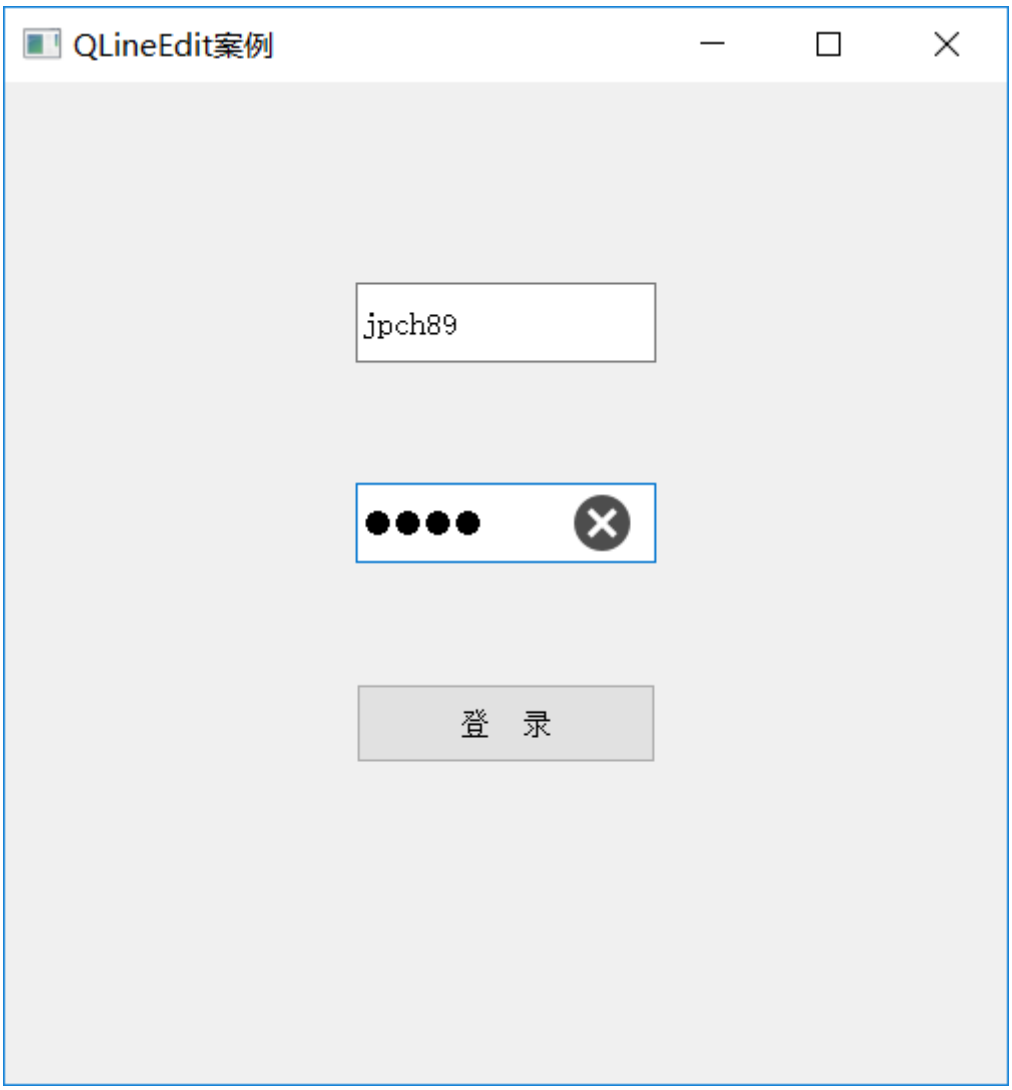
```
if __name__ == '__main__':
    import sys

    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

120. Python-GUI编程-PyQt5-QLineEdit-清空按钮的显示


```
# 拓展1：假如账号和密码其中之一没有内容，登录按钮不可用
# 拓展2：账号下面和密码下面都有提示标签，一开始是隐藏的，根据输入展示提示信息
# 拓展3：账号和密码是业务逻辑，需要与界面分离

from PyQt5.Qt import *


class AccountTool:  # 默认继承 object
    ACCOUNT_ERROR = 1
    PWD_ERROR = 2
    SUCCESS = 3

    @staticmethod
    def check_login(account, pwd):
        # 把账号和密码发送给服务器，等待服务器返回结果
        if account != 'jpch89':
            # 返回一个状态
            return AccountTool.ACCOUNT_ERROR
            # 或者通过类名访问也可以
            # return AccountTool.ACCOUNT_ERROR
        if pwd != '666':
            return AccountTool.PWD_ERROR
        return AccountTool.SUCCESS
```

```
class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.pwd_le.setEchoMode(QLineEdit.Password)
        self.login_btn = QPushButton(self)
        self.login_btn.setText('登  录')

        # 占位文本的提示
        self.account_le.setPlaceholderText('请输入账号')
        self.pwd_le.setPlaceholderText('请输入密码')

        # 设置密码文本框，自动显示清空按钮
        self.pwd_le.setClearButtonEnabled(True)
        # 获取清空按钮是否可用
        print('密码框清空按钮是否可用：', self.pwd_le.isClearButtonEnabled())

        # 连接槽函数
        self.login_btn.clicked.connect(self.login)

    def login(self):
        # 获取账号和密码信息
        account = self.account_le.text()
        pwd = self.pwd_le.text()
        print(f'账号：{account}\n密码：{pwd}')
        state = AccountTool.check_login(account, pwd)
        if state == AccountTool.ACCOUNT_ERROR:
            print('账号错误')
            self.account_le.setText('')
            self.pwd_le.setText('')
            self.account_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        if state == AccountTool.PWD_ERROR:
            print('密码错误')
            self.pwd_le.setText('')
            self.pwd_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
```

```
                return None

            # 这里可以不用判定!
            if state == AccountTool.SUCCESS:
                print('登录成功')

        def resizeEvent(self, evt):
            widget_w = 150
            widget_h = 40
            margin = 60
            self.account_le.resize(widget_w, widget_h)
            self.pwd_le.resize(widget_w, widget_h)
            self.login_btn.resize(widget_w, widget_h)

            x = (self.width() - widget_w) / 2

            self.account_le.move(x, self.height() / 5)
            self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
            self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)


    if __name__ == '__main__':
        import sys

        app = QApplication(sys.argv)

        window = Window()
        window.show()
        sys.exit(app.exec_())
```



121. Python-GUI编程-PyQt5-QLineEdit-添加自定义行为

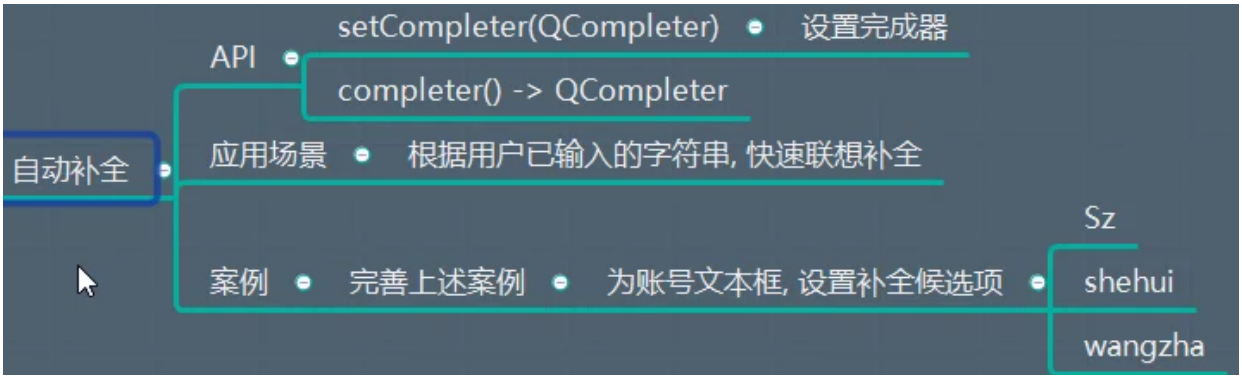

```python
# 拓展1：假如账号和密码其中之一没有内容，登录按钮不可用
# 拓展2：账号下面和密码下面都有提示标签，一开始是隐藏的，根据输入展示提示信息
# 拓展3：账号和密码是业务逻辑，需要与界面分离

from PyQt5.Qt import *


class AccountTool:  # 默认继承 object
```

```
    ACCOUNT_ERROR = 1
    PWD_ERROR = 2
    SUCCESS = 3

    @staticmethod
    def check_login(account, pwd):
        # 把账号和密码发送给服务器，等待服务器返回结果
        if account != 'jpch89':
            # 返回一个状态
            return AccountTool.ACCOUNT_ERROR
            # 或者通过类名访问也可以
            # return AccountTool.ACCOUNT_ERROR
        if pwd != '666':
            return AccountTool.PWD_ERROR
        return AccountTool.SUCCESS


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.pwd_le.setEchoMode(QLineEdit.Password)
        self.login_btn = QPushButton(self)
        self.login_btn.setText('登　录')

        # 占位文本的提示
        self.account_le.setPlaceholderText('请输入账号')
        self.pwd_le.setPlaceholderText('请输入密码')

        # 设置密码文本框，自动显示清空按钮
        self.pwd_le.setClearButtonEnabled(True)

        # 添加自定义动作（明文和密文切换）
        # 三种写法：
        # addAction(self, QAction)
        # addAction(self, QAction, QLineEdit.ActionPosition)
        # addAction(self, QIcon, QLineEdit.ActionPosition) -> QAction
        # 设置文本框作为它的父对象，文本框没有的时候，它就会被自动释放掉
        action = QAction(self.pwd_le)
        action.setIcon(QIcon('img/隐藏密码.png'))

        def change():
            # print('切换明文和密文')
            if self.pwd_le.echoMode() == QLineEdit.Password:
```

```
                self.pwd_le.setEchoMode(QLineEdit.Normal)
                action.setIcon(QIcon('img/显示密码.png'))
            else:
                self.pwd_le.setEchoMode(QLineEdit.Password)
                action.setIcon(QIcon('img/隐藏密码.png'))

        action.triggered.connect(change)
        # 图标在尾部
        self.pwd_le.addAction(action, QLineEdit.TrailingPosition)
        # 图标在头部
        # self.pwd_le.addAction(action, QLineEdit.LeadingPosition)

        # 连接槽函数
        self.login_btn.clicked.connect(self.login)

    def login(self):
        # 获取账号和密码信息
        account = self.account_le.text()
        pwd = self.pwd_le.text()
        print(f'账号：{account}\n密码：{pwd}')
        state = AccountTool.check_login(account, pwd)
        if state == AccountTool.ACCOUNT_ERROR:
            print('账号错误')
            self.account_le.setText('')
            self.pwd_le.setText('')
            self.account_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        if state == AccountTool.PWD_ERROR:
            print('密码错误')
            self.pwd_le.setText('')
            self.pwd_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        # 这里可以不用判定！
        if state == AccountTool.SUCCESS:
            print('登录成功')

    def resizeEvent(self, evt):
        widget_w = 200
        widget_h = 40
        margin = 60
        self.account_le.resize(widget_w, widget_h)
        self.pwd_le.resize(widget_w, widget_h)
        self.login_btn.resize(widget_w, widget_h)
```

```
        x = (self.width() - widget_w) / 2

        self.account_le.move(x, self.height() / 5)
        self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
        self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)


if __name__ == '__main__':
    import sys

    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

122. Python-GUI编程-PyQt5-QLineEdit-自动补全联想



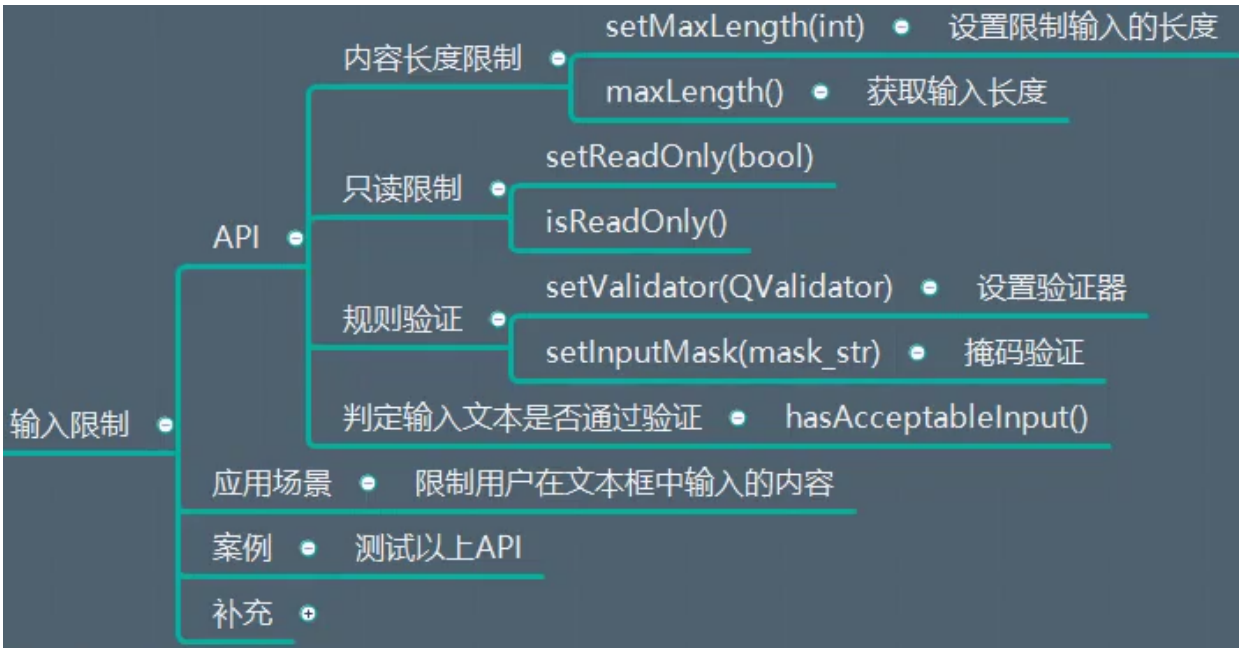

```
# 拓展1：假如账号和密码其中之一没有内容，登录按钮不可用
# 拓展2：账号下面和密码下面都有提示标签，一开始是隐藏的，根据输入展示提示信息
# 拓展3：账号和密码是业务逻辑，需要与界面分离

from PyQt5.Qt import *


class AccountTool:  # 默认继承 object
    ACCOUNT_ERROR = 1
    PWD_ERROR = 2
    SUCCESS = 3

    @staticmethod
    def check_login(account, pwd):
        # 把账号和密码发送给服务器，等待服务器返回结果
        if account != 'jpch89':
```

```
            # 返回一个状态
            return AccountTool.ACCOUNT_ERROR
            # 或者通过类名访问也可以
            # return AccountTool.ACCOUNT_ERROR
        if pwd != '666':
            return AccountTool.PWD_ERROR
        return AccountTool.SUCCESS


class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit案例')
        self.resize(500, 500)
        # 设置最小尺寸
        self.setMinimumSize(400, 400)
        self.setup_ui()

    def setup_ui(self):
        self.account_le = QLineEdit(self)
        self.pwd_le = QLineEdit(self)
        self.pwd_le.setEchoMode(QLineEdit.Password)
        self.login_btn = QPushButton(self)
        self.login_btn.setText('登  录')

        # 占位文本的提示
        self.account_le.setPlaceholderText('请输入账号')
        self.pwd_le.setPlaceholderText('请输入密码')

        # 设置密码文本框，自动显示清空按钮
        self.pwd_le.setClearButtonEnabled(True)

        # 添加自定义动作（明文和密文切换）
        # 三种写法:
        # addAction(self, QAction)
        # addAction(self, QAction, QLineEdit.ActionPosition)
        # addAction(self, QIcon, QLineEdit.ActionPosition) -> QAction
        # 设置文本框作为它的父对象，文本框没有的时候，它就会被自动释放掉
        action = QAction(self.pwd_le)
        action.setIcon(QIcon('img/隐藏密码.png'))

        def change():
            # print('切换明文和密文')
            if self.pwd_le.echoMode() == QLineEdit.Password:
                self.pwd_le.setEchoMode(QLineEdit.Normal)
                action.setIcon(QIcon('img/显示密码.png'))
            else:
                self.pwd_le.setEchoMode(QLineEdit.Password)
                action.setIcon(QIcon('img/隐藏密码.png'))

        action.triggered.connect(change)
        # 图标在尾部
```

```
        self.pwd_le.addAction(action, QLineEdit.TrailingPosition)
        # 图标在头部
        # self.pwd_le.addAction(action, QLineEdit.LeadingPosition)

        # 自动补全
        # QCompleter(Iterable[str], parent: QObject = None)
        # 完成器生命周期归账号文本框 account_le 管理
        completer = QCompleter(['jpch89', '进击的团子', 'jinjidetuanzi'], s
elf.account_le)
        self.account_le.setCompleter(completer)

        # 连接槽函数
        self.login_btn.clicked.connect(self.login)

    def login(self):
        # 获取账号和密码信息
        account = self.account_le.text()
        pwd = self.pwd_le.text()
        print(f'账号：{account}\n密码：{pwd}')
        state = AccountTool.check_login(account, pwd)
        if state == AccountTool.ACCOUNT_ERROR:
            print('账号错误')
            self.account_le.setText('')
            self.pwd_le.setText('')
            self.account_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        if state == AccountTool.PWD_ERROR:
            print('密码错误')
            self.pwd_le.setText('')
            self.pwd_le.setFocus()
            # 可写可不写
            # 但是写了可以跳过后续判定
            # 加快程序执行
            return None

        # 这里可以不用判定！
        if state == AccountTool.SUCCESS:
            print('登录成功')

    def resizeEvent(self, evt):
        widget_w = 200
        widget_h = 40
        margin = 60
        self.account_le.resize(widget_w, widget_h)
        self.pwd_le.resize(widget_w, widget_h)
        self.login_btn.resize(widget_w, widget_h)

        x = (self.width() - widget_w) / 2
```

```
        self.account_le.move(x, self.height() / 5)
        self.pwd_le.move(x, self.account_le.y() + widget_h + margin)
        self.login_btn.move(x, self.pwd_le.y() + widget_h + margin)


if __name__ == '__main__':
    import sys

    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

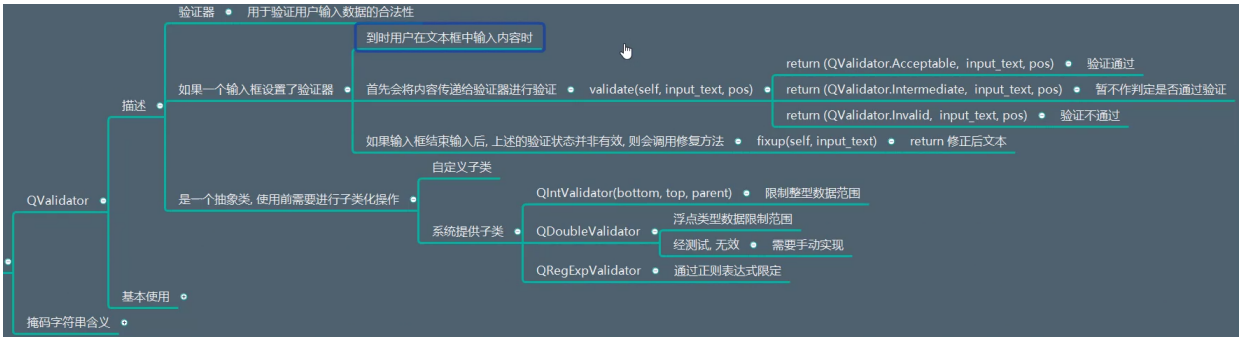
123. Python-GUI编程-PyQt5-QLineEdit-文本内容限制-长度和只读限制

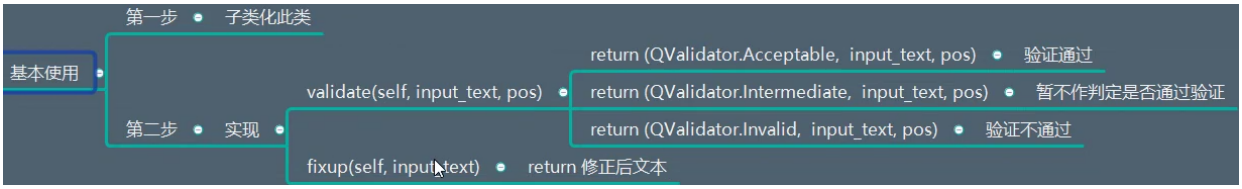



```
# 0. 导入需要的包和模块
import sys
from PyQt5.Qt import *

# 1. 创建一个应用程序对象
app = QApplication(sys.argv)

# 2. 控件的操作
```

```
    # 2.1 创建控件
    window = QWidget()
    # 2.2 设置控件
    window.setWindowTitle('输入限制')
    window.resize(500, 500)

    le_a = QLineEdit(window)
    le_a.move(100, 100)
    # 长度限制
    le_a.setMaxLength(3)
    # 获取长度限制
    print('上面的文本框的长度限制：', le_a.maxLength())
    # 注意：通过代码设置文本，也会自动截断到 3 个字符长度

    le_b = QLineEdit(window)
    le_b.move(100, 200)
    # 只读限制
    le_b.setReadOnly(True)
    # 获取是否只读
    print('下面的文本框是否只读：', le_b.isReadOnly())
    # 可以通过代码设置文本
    le_b.setText('只能看，不能改')

    # 2.3 展示控件
    window.show()

    # 3. 应用程序的执行，进入到消息循环
    sys.exit(app.exec_())
```

124. Python-GUI编程-PyQt5-QLineEdit-文本内容限制-验证器的使用





125. Python-GUI编程-PyQt5-QLineEdit-文本内容限制-验证器案例-方式1

```
from PyQt5.Qt import *


class AgeValidator(QValidator):
    def validate(self, input_str, pos_int):
        print(input_str, pos_int)

        # 判定
        # 结果字符串，应该全部都是由一些数字组成
        # 方法1：正则
        # 方法2：try except
        # 自己想的方法3：str.isdigits()
        try:
            if 18 <= int(input_str) <= 180:
                return QValidator.Acceptable, input_str, pos_int
            elif 1 <= int(input_str) <= 17:
                return QValidator.Intermediate, input_str, pos_int
            else:
                # 注意：无效数据不会被展示在文本框内部！
                return QValidator.Invalid, input_str, pos_int
        except:
            if len(input_str) == 0:
                return QValidator.Intermediate, input_str, pos_int
            return QValidator.Invalid, input_str, pos_int

        # 这样返回，不管怎么样都会被重置为 测试一下
        # return QValidator.Acceptable, '测试一下', 2

    def fixup(self, p_str):
        print('fixup:', p_str)
        try:
            if int(p_str) < 18:
                return '18'
            return '180'
        except:
            return '18'

class Window(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit验证器')
        self.resize(500, 500)
```

```
        self.setup_ui()

    def setup_ui(self):
        le = QLineEdit(self)
        le.move(100, 100)

        # 验证要求：数字 18-180
        # QValidator 是 C++ 抽象类，不能直接被实例化
        # 所以要对其进行子类化
        validator = AgeValidator()
        le.setValidator(validator)

        le2 = QLineEdit(self)
        le2.move(100, 200)


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

126. Python-GUI编程-PyQt5-QLineEdit-文本内容限制-验证器案例-方式2

```
from PyQt5.Qt import *


class AgeValidator(QIntValidator):
    def fixup(self, p_str):
        # 打印出结果，意味着数据是无效的
        # 需要进行修复
        print('fixup: ', p_str)
        # 假如 p_str 长度等于零，不会进行 int(p_str) 类型转换
        # 短路逻辑！
        # 注意：这两个条件反过来就不行了！
        if len(p_str) == 0 or int(p_str) < 18:
            return '18'


class Window(QWidget):
```

```
    def __init__(self):
        super().__init__()
        self.setWindowTitle('QLineEdit验证器')
        self.resize(500, 500)
        self.setup_ui()

    def setup_ui(self):
        le = QLineEdit(self)
        le.move(100, 100)

        # 验证要求：数字 18-180
        # 只能限定最大值，不能限定最小值
        # 中间状态没有去处理
        # 所以可以借助它，在其基础上进行二次定制
        validator = AgeValidator(18, 180)
        le.setValidator(validator)

        le2 = QLineEdit(self)
        le2.move(100, 200)


if __name__ == '__main__':
    import sys
    app = QApplication(sys.argv)

    window = Window()
    window.show()
    sys.exit(app.exec_())
```

127.