

Size

期望尺寸: `.sizeHint()`, 获取宽高 `.sizeHint().height()`, `.sizeHint().width()`

最小期望尺寸: `minimumSizeHint()`

widget

```
widget.x()

widget.y()

widget.width()

widget.height()

widget.geometry().\ widget.frameGeometry().

setIcon(QIcon())

setWindowIcon()

setWindowTitle()

QToolTip.setFont(QFont(...))
```

`self.sender()`

QLabel

常用信号:

- `linkHovered`
- `linkActivated`

`setAlignment()`

`setText()`

`setColor()`

`setPalette()`

`setToolTip()`

`setPixmap()`

`setOpenExternalLinks(True)`: 打开链接

`setBuddy()`: 设置伙伴关系

`setColor()`

`setAutoFillBackground(True)`

Layout

`addWidget(控件, 起始行, 起始列, 所占行数, 所占列数)`

`formLayout.addRow()`

QLineEdit

`setPlaceholderText()`: 未输入时显示的文字

`setValidator()`

`setInputMask()`: 掩码

`setMaxLength(4)`

`setFont(QFont('Arial', 20))`

`setReadOnly()`

信号

textChanged

editingFinished

EchoMode

setEchoMode(QLineEdit.Normal)

1. Normal
2. NoEcho
3. Password
4. PasswordEchoEdit

校验器

QIntValidator、QDoubleValidator、QRegExpValidator

```
from PyQt5.QtCore import QRegExp
```

```
validator.setRange()
```

```
doubleValidator.setDecimal(2) # 精度
```

```
regValidator.setRegExp(QRegExp(...))
```

```
DoubleValidator(0.99, 99.99, 2)
```

掩码

掩码字符

说明

A	ASCII字母字符(A-Z、a-z) 是必须输入的
a	ASCII字母字符(A-Z、a-z) 是允许输入的,但不是必需输入的
N	ASCII字母字符(A-Z、a-z、0-9)是必须输入的
n	ASCII字母字符(A-Z、a-z、0-9)是允许输入的,但不是必需输入的
X	任何字符都是必须输入的
x	任何字符都是允许输入的,但不是必需输入的
9	ASCII数字字符(0-9)是必须输入的
0	ASCII数字字符(0-9)是允许输入的,但不是必需输入的
D	ASCII数字字符(1-9)是必须输入的
d	ASCII数字字符(1-9)是允许输入的,但不是必需输入的
#	ASCII数字字符或加减符号是允许输入的,但不是必需输入的
H	十六进制格式字符(A-F、a-f、0-9) 是必须输入的
h	十六进制格式字符(A-F、a-f、0-9) 是允许输入的,但不是必需输入的
B	二进制格式字符(0,1)是必须输入的
b	二进制格式字符(0,1)是允许输入的,但不是必需输入的
>	所有的字母字符都大写
<	所有的字母字符都小写
!	关闭大小写转换
**	使用""转义上面列出的字符

;, #, 填充

QTextEdit

setPlainText()

setHtml()

toHtml()

toText()

```
self.button1.clicked.connect(lambda:self.whichButton(参数))
```

RadioButton

```
radioButton.setChecked(True)
```

QCheckBox

3 kind of state

```
checkboxbox.setTristate(True)
```

```
checkboxbox.setCheckState(Qt.PartiallyChecked)
```

```
checkboxbox.checkState
```

信号

```
stateChanged
```

QComboBox

```
.addItem()
```

```
.addItems([])
```

信号

```
currentIndexChanged
```

QSlider

```
QSlider(Qt.Horizontal)
```

```
slider.setMinimum()
```

```
slider.setMaximum()
```

```
slider.setSingleStep()
```

```
slider.setValue()
```

```
slider.setTickPosition(QSlider.TicksBelow) 刻度显示在下方
```

```
slider.setTickInterval() 刻度的间隔
```

信号

```
valueChanged
```

QSpinBox

```
setValue
```

信号

```
valueChanged
```

Dialog

```
setWindowModality
```

QMessageBox

```
.about(self, "关于", "这是一个关于对话框")
```

```
reply = box.information(self, "消息", "消息对话框", QMessageBox.Yes|QMessageBox.No, QMessageBox.Yes)
```

```
.warning()
```

```
.critical()
```

```
.question()
```

QInputDialog

```
QInputDialog.getInt()
```

```
.getText()
```

```
.getItem()
```

eg

```
items = ('A', 'B', 'C')
item, ok = QInputDialog.getItem(self, '请选择', '列表', items)
```

QFileDialog

```
fname, _ = QFileDialog.getOpenFileName(self, '打开文件', '.', '图像文件 (*.jpg *.png)')
```

```
self.imageLabel.setPixmap(fname)
```

```
dialog = QFileDialog()
dialog.setFileMode(QFileDialog.AnyFile)
dialog.setFilter(QDir.Files)
```

QPainter

```
painter = QPainter()
```

```
painter.begin()
```

```
painter.drawText(区域, 对齐方式, 文本)
```

```
painter.end()
```

```
painter.setPen(QColor(...))
painter.setFont(QFont(...))
painter.drawText(event.rect(), Qt.AlignCenter, self.text)
```

```
QPen(Qt.Red, Qt.solidLine)
```

```
painter.drawLine(20, 40, 250, 80)
```

```
pen.setStyle(Qt.DashDotDotLine)
```

```
pen.setStyle(Qt.CustomDashLine)
```

```
pen.setDashPattern([1, 4, 5, 4])
```

```
rect = QRect(0, 10, 100, 100)
qpainter.drawArc(rect, 0, 50 * 16) # 一个alen相等于1/16度
qpainter.drawChord(...)
qpainter.drawPie(...)
qpainter.drawEllipse(...)
```

多边形

```
point1 = QPoint(140, 100)
point2 = QPoint(...)
...
polygon = QPolygon(point1, point2, ...)
qpainter.drawPolygon(polygon)
```

图像

```
image = QImage('...')
qpainter.drawImage(...)
```

QBrush

```
qpainter.setBrush(QBrush(Qt.SolidPattern))
```

```
qpainter.drawRect(...)
```

拖拽

```
setAcceptDrops(True)
```

dragEnterEvent: 拖到区域时触发

dropEvent: 放下时触发

setDragEnable(True): 使可拖动

剪贴板

```
clipboard = QApplication.clipboard()
clipboard.setText('...')
clipboard.setPixmap(QPixmap('...'))
mimeData = QMimeData()
mimeData.setHtml('...')
clipboard.setMimeData(mimeData)

url = QUrl.fromLocalFile('F:\\chrome浏览器下载\\Document\\pandas.pdf')
data = QMimeData()
data.setUrls([url])
app.clipboard().setMimeData(data)
```

日历控件

```
cal = QCalendarWidget(self)

cal.setMinimumDate(QDate(1988, 1, 1))

cal.setMaximumDate()

cal.setGridVisible(True)

cal.selectedDate()
```

QDateTimeEdit

```
dateTimeEdit1 = QDateTimeEdit()

dateTimeEdit2 = QDateTimeEdit()

dateEdit = QDateTimeEdit(QDate.currentDate())

timeEdit = QDateTimeEdit(QTime.currentTime())

dateTimeEdit1.setDisplayFormat('yyyy-MM-dd HH:mm:ss')

dateEdit.setDisplayFormat('yyyy.MM.dd')

timeEdit.setDisplayFormat("HH:mm:ss")

edit.datetime()
```

菜单栏

```
bar = self.menuBar() # 获取菜单栏
file = bar.addMenu('文件')
file.addAction("新建")
save = QAction("保存", self)
save.setShortcut('Ctrl+S')
file.addAction(save)
```

工具栏

```
tbar = self.addToolBar("File")
new = QAction(QIcon('QQ (1).png'), "new", self)
tbar.addAction(new)
# tbar.setToolButtonStyle(Qt.ToolButtonIconOnly)
# tbar.setToolButtonStyle(Qt.ToolButtonTextUnderIcon)
# tbar.setToolButtonStyle(Qt.ToolButtonTextOnly)
tbar.setToolButtonStyle(Qt.ToolButtonTextBesideIcon)
```

信号

```
actionTriggered
```

状态栏

```
statusBar

self.statusBar = QStatusBar()
self.setStatusBar(self.statusBar)
```

打印机

```

editor = QTextEdit("Hello", self)
    printer = QtPrintSupport.QPrinter()
    painter = QPainter()
    painter.begin(printer) # 重定向
    screen = editor.grab()
    painter.drawPixmap(10, 10, screen)
    painter.end()

printDialog = QPageSetupDialog(self.printer, self)

printDialog.exec()

printDialog = QPrintDialog(self.printer, self)
if QDialog.Accepted == printDialog.exec():
    self.editor.print(self.printer)

```

QTableView

```

self.model = QStandardItemModel(4, 3)
self.model.setHorizontalHeaderLabels(['id', '姓名', '年龄'])

self.tableView = QTableView(self)
self.tableView.setModel(self.model)
item = QStandardItem('10')
item2 = QStandardItem('雷神')
item3 = QStandardItem('2000')
self.model.setItem(2, 0, item)
self.model.setItem(2, 1, item2)
self.model.setItem(2, 2, item3)
layout = QVBoxLayout()
layout.addWidget(self.tableView)
self.setLayout(layout)
self.setGeometry(100, 100, 500, 500)

```

在QWidget

ListView

```

listmodel = QStringListModel()

listmodel.setStringList()

listview.setModel(listmodel)

```

QListWidget

```
listwidget.addItem(...)
```

信号

```
itemClicked
```

QTableWidget

```

tablewidget.setRowCount()

tablewidget.setColumnCount()

.setHorizontalHeaderLabels()

.setItem(QTableWidgetItem())

.setEditTriggers(QAbstractItemView.NoEditTriggers)

.setSelectionBehavior(QAbstractView.SelectRows)

.resizeColumnsToContents()

.resizeRowsToContents()

.horizontalHeader().setVisible(False)

.setShowGrid(False)

.setCellWidget() # 添加控件

tablewidget.findItems(text, Qt.MatchExactly)

```

```

tableWidget.verticalScrollBar().setSliderPosition(row)

item.setFont()

item.setForeground(QBrush(QColor(...)))

item.setBackground()

tableWidget.sortItems(列数, orderType)

item.setTextAlignment(Qt.AlignRight | Qt.AlignBottom)

tableWidget.setSpan(行索引, 列索引, 行数, 列数)

tableWidget.setRowHeight(行索引, 高度)

tableWidget.setColumnWidth()

QTableWidgetItem(QIcon('...'), '...')

tableWidget.setIconSize(QSize(宽度, 高度))

```

上下文菜单

信号

```

customContextMenuRequested

    self.tableWidget = QTableWidgetItem()
    self.tableWidget.setContextMenuPolicy(Qt.CustomContextMenu)
    self.tableWidget.customContextMenuRequested.connect(self.generateMenu)
    self.tableWidget.setRowCount(3)
    self.tableWidget.setColumnCount(4)
    item = QTableWidgetItem("Hello")
    self.tableWidget.setItem(1, 2, item)
    layout = QVBoxLayout()
    layout.addWidget(self.tableWidget)

    self.setLayout(layout)
    self.setGeometry(100, 100, 500, 500)
    self.show()

def generateMenu(self, pos): # pos是位置
    menu = QMenu()
    item1 = menu.addAction('菜单项1')
    item2 = menu.addAction('菜单项2')
    item3 = menu.addAction('菜单项3')
    # 被阻塞
    screenPos = self.tableWidget.mapToGlobal(pos)

    action = menu.exec(screenPos)

```

树控件

```

self.tree = QTreeWidget()
self.tree.setColumnCount(2) # 列数
self.tree.setHeaderLabels(['Hello', 'World'])
root = QTreeWidgetItem(self.tree)
root.setText(0, "root")
self.tree.setColumnWidth(0, 120)
child1 = QTreeWidgetItem(root)
child1.setText(0, 'son')
child1.setCheckState(0, Qt.Checked)
self.tree.expandAll()

clicked会传入一个index

item = self.tree.currentItem()

node = QTreeWidgetItem(item)

node.setText()

root = self.tree.incisibleRootItem()
for item in self.tree.selectedItems():
    (item.parent() or root).removeChild(item)

```

QTreeView

```
model = QDirModel()
tree = QTreeView()
tree.setModel(model)
```

QTabWidget

```
class Example(QTabWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        layout = QVBoxLayout()
        # model = QDirModel()
        # tree = QTreeView()
        # tree.setModel(model)
        self.tab1 = QWidget()
        self.tab2 = QWidget()
        self.tab3 = QWidget()

        self.addTab(self.tab1, '选项卡1')
        self.addTab(self.tab2, '选项卡2')
        self.addTab(self.tab3, '选项卡3')

        self.setLayout(layout)
        # layout.addWidget(tree)
        self.setGeometry(100, 100, 500, 500)
        self.show()
```

堆栈控件

```
self.list = QListWidget()
self.list.insertItems(0, ['hads', 'ndajf', 'asdf'])
self.stack1 = QWidget()
self.stack2 = QWidget()
self.stack3 = QWidget()
self.stack = QStackedWidget()
self.stack.addWidget(self.stack1)
self.stack.addWidget(self.stack2)
self.stack.addWidget(self.stack3)
self.setLayout(layout)
layout.addWidget(self.list)
layout.addWidget(self.stack)
```

```
list.currentRowChanged.connect()
```

停靠控件

```
self.items = QDockWidget('Dockable', self)
self.listWidget = QListWidget()
self.listWidget.addItem(['item1', 'item2', 'item3'])
self.items.setWidget(self.listWidget)
self.setCentralWidget(QLineEdit())
self.addDockWidget(Qt.RightDockWidgetArea, self.items)
self.items.setFloating(True)
```

多文本窗口

```
sub = QMdiSubWindow()

sub.setWidget(QTextEdit())

self.mdi.addSubWindow(sub)

self.mid.cascadeSubWindows()

self.mid.tileSubWindows()
```

滚动条控件

信号

```
sliderMoved
```

多线程


```

class Example(QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.label = QLabel("显示当前时间")
        self.startBtn = QPushButton('开始')
        self.endBtn = QPushButton('结束')
        layout = QGridLayout()

        self.timer = QTimer()
        self.timer.timeout.connect(self.showTime)

        layout.addWidget(self.label, 0, 0, 1, 2)
        layout.addWidget(self.startBtn, 1, 0)
        layout.addWidget(self.endBtn, 1, 1)
        self.startBtn.clicked.connect(self.StartTimer)
        self.endBtn.clicked.connect(self.endTimer)

        self.setLayout(layout)
        self.show()

    def showTime(self):
        time = QDateTime.currentDateTime()

        timeDisplay = time.toString("yyyy-MM-dd hh:mm:ss dddd")
        self.label.setText(timeDisplay)

    def StartTimer(self):
        self.timer.start(1000) # 每隔一秒
        self.startBtn.setEnabled(False)
        self.endBtn.setEnabled(True)

    def endTimer(self):
        self.timer.stop()
        self.startBtn.setEnabled(True)
        self.endBtn.setEnabled(False)

```

让程序定时关闭

```

label = QLabel("HELLO")
label.setWindowFlags(Qt.SplashScreen | Qt.FramelessWindowHint)
label.show()
QTimer.singleShot(5000, app.quit)

```

```

import sys

```

```

from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5.QtWidgets import *

```

```

sec = 0

```

```

class WorkThread(QThread):
    timer = pyqtSignal() # 每隔一秒发送一次信号
    end = pyqtSignal() # 计数完成后发送一次信号

    def run(self):
        while True:
            self.sleep(1) # 休眠1秒
            if sec == 5:
                self.end.emit() # 发送end信号
                break
            self.timer.emit() # 发送timer信号

```

```

class Counter(QWidget):
    def __init__(self):
        super(Counter, self).__init__()
        layout = QVBoxLayout()
        self.lcdNumber = QLCDNumber()
        layout.addWidget(self.lcdNumber)
        button = QPushButton('开始计数')
        layout.addWidget(button)
        self.workThread = WorkThread()
        self.workThread.timer.connect(self.countTime)
        self.workThread.end.connect(self.end)
        button.clicked.connect(self.work)
        self.setLayout(layout)
        self.show()

    def countTime(self):

```

```

        global sec
        sec += 1
        self.lcdNumber.display(sec)

    def end(self):
        QMessageBox.information(self, '消息', '计数结束', QMessageBox.Ok)

    def work(self):
        self.workThread.start()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Counter()
    sys.exit(app.exec_())

```

Web

```

import sys
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtWebEngineWidgets import *

class WebEngineView(QMainWindow):
    def __init__(self):
        super(WebEngineView, self).__init__()
        self.setGeometry(5, 30, 1355, 730)

        self.browser = QWebEngineView()
        self.browser.load(QUrl('https://www.baidu.com'))
        self.setCentralWidget(self.browser)
        self.show()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = WebEngineView()
    sys.exit(app.exec_())

```

本地

```

self.browser = QWebEngineView()
url = os.getcwd() + '/1.html'
self.browser.load(QUrl.fromLocalFile(url))
self.setCentralWidget(self.browser)
self.show()

```

嵌入

```

class WebEngineView(QMainWindow):
    def __init__(self):
        super(WebEngineView, self).__init__()
        self.setGeometry(5, 30, 1355, 730)

        self.browser = QWebEngineView()
        self.browser.setHtml('''
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Hello</title>
</head>
<body>
<h1>Title</h1>
</body>
</html>
''')
        self.setCentralWidget(self.browser)
        self.show()

setSpacing()

addStretch()

QGridLayout

addWidget()

```

```
formlayout.addRow()
```

QSplitter

```
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class MySig(QObject):
    sendmsg = pyqtSignal(object)

    def run(self):
        self.sendmsg.emit('Hello world')

class MySlot(QObject):
    def get(self, msg):
        print(msg)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    send = MySig()
    slot = MySlot()
    send.sendmsg.connect(slot.get)
    send.run()
    send.sendmsg.disconnect(slot.get) # 断开
    sys.exit(app.exec_())
```

发送含有多个参数的信号

```
sendmsg = pyqtSignal(str, int, int)

sendmsg.emit('', 1, 1)

# 声明一个重载版本的信号，也就是槽函数的参数可以是int和str类型，也可以只有一个str类型的参数
signal6 = pyqtSignal([int,str],[str])
```

关联到第二个重载形式，`signal[str].connect()`

发送时发送和调用都要指定参数类型

```
class AutoSignalSlot(QWidget):
    def __init__(self):
        super(AutoSignalSlot, self).__init__()
        self.okButton = QPushButton("ok", self)
        self.okButton.setObjectName("okButton")
        self.okButton1 = QPushButton("cancel", self)
        self.okButton1.setObjectName("cancelButton")
        layout = QHBoxLayout()
        layout.addWidget(self.okButton)
        self.setLayout(layout)
        QtCore.QMetaObject.connectSlotsByName(self)
        #self.okButton.clicked.connect(self.on_okButton_clicked)

    @QtCore.pyqtSlot()
    def on_okButton_clicked(self):
        print("点击了ok按钮")

    @QtCore.pyqtSlot()
    def on_cancelButton_clicked(self):
        print("点击了cancel按钮")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    example = AutoSignalSlot()
    example.show()
    sys.exit(app.exec_())
```

```
clicked.connect(lambda: self.onButtonClick(参数))
```

```
from functools import partial
```

```
clicked.connect(partial(self.onButtonClick, ))
```

窗口控件风格

windows, fusion、macintosh

```

from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
from PyQt5.QtGui import *
import sys

class Example(QWidget):
    def __init__(self):
        super(Example, self).__init__()
        self.initUI()

    def initUI(self):
        layout = QHBoxLayout()
        self.styleLabel = QLabel('风格')
        self.styleComobox = QComboBox()
        self.styleComobox.addItem(QStyleFactory.keys())
        index = self.styleComobox.findText(QApplication.style().objectName(), Qt.MatchFixedString)
        print(QApplication.style().objectName())
        self.styleComobox.setCurrentIndex(index)
        self.styleComobox.activated[str].connect(self.handleC)
        layout.addWidget(self.styleLabel)
        layout.addWidget(self.styleComobox)
        self.setLayout(layout)
        self.setGeometry(100, 100, 500, 500)
        self.show()

    def handleC(self, style):
        print(style)
        QApplication.setStyle(style)

if __name__ == '__main__':
    app = QApplication(sys.argv)
    ex = Example()
    sys.exit(app.exec_())

```

设置窗口风格

```

setWindowFlags

desktop = QApplication.desktop()
# 获取桌面可用尺寸
rect = desktop.availableGeometry()

self.setGeometry(rect)

```

QSS

```

setProperty('name', 'btn')

QPushButton[name='btn2']{
    background-color: red;
    height: 120;
    font-size: 60px;
}

setObjectName('myComoBox')

QComboBox#myComoBox::drop-down{
    image:url(...)
}

event.globalPos() # 当前点相对屏幕的位置

event.pos() # 相对窗口的位置，不包含标题栏

self.pos() # 包含标题栏

movie = QMovie('filepath')

movie.start()

img = QImage(filename)

img.scaled(width, height, flags)

setWindowOpacity() # 透明度

QPropertyAnimation

```

```
'''
```

用动画效果改变窗口尺寸

QPropertyAnimation

```
'''
```

```
from PyQt5.QtGui import *
from PyQt5.QtCore import *
from PyQt5.QtWidgets import *
import sys
```

```
class AnimWindow(QWidget):
    def __init__(self):
        super(AnimWindow, self).__init__()
        self.OrigHeight = 50
        self.ChangeHeight = 150
        self.setGeometry(QRect(500, 400, 150, self.OrigHeight))
        self.btn = QPushButton('展开', self)
        self.btn.setGeometry(10, 10, 60, 35)
        self.btn.clicked.connect(self.change)
    def change(self):
        currentHeight = self.height()
        if self.OrigHeight == currentHeight:
            startHeight = self.OrigHeight
            endHeight = self.ChangeHeight
            self.btn.setText('收缩')
        else:
            startHeight = self.ChangeHeight
            endHeight = self.OrigHeight
            self.btn.setText('展开')

        self.animation = QPropertyAnimation(self, b'geometry')
        self.animation.setDuration(500)
        self.animation.setStartValue(QRect(500, 400, 150, startHeight))
        self.animation.setEndValue(QRect(500, 400, 150, endHeight))
        self.animation.start()

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = AnimWindow()
    window.show()
    sys.exit(app.exec_())
```