

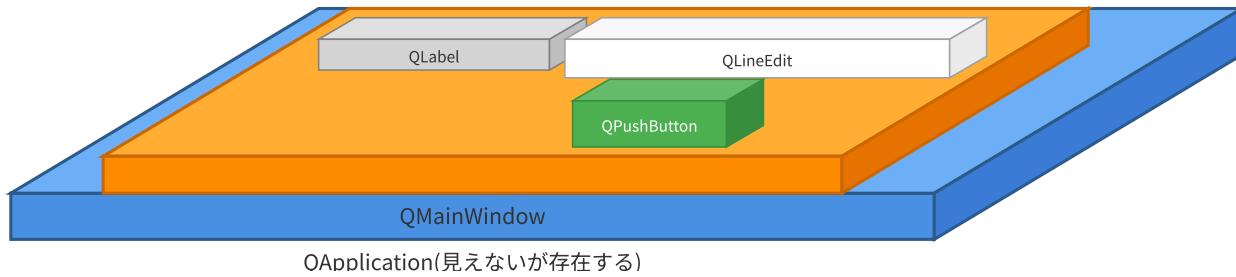
PySide6の動作の仕組み

QApplication と QWidget

PySide6アプリケーションは、**QApplication**と**QWidget**という2つのクラスのオブジェクトを積み上げて作ります。

クラス	説明
QApplication	アプリケーション全体を管理するクラス。イベントループの実行、システム設定の管理、全ウィ젯の統括を担当します。アプリケーション内で1つだけ生成して利用します。
QWidget	すべてのUI要素の基底クラス。ウィンドウやボタン、ラベルなどのUIコンポーネントは全てQWidgetを継承しています。親子関係による階層構造を形成し、効率的なメモリ管理とイベント処理を実現します。

「QApplication の上に QWidget が乗っている。QWidget の上にもさらに QWidget が乗っている」というのが、PySide6のUIのイメージです。



上の図で、QMainWindow, QLabel, QLineEdit, QPushButton はいずれも QWidget クラスのサブクラス(QWidgetクラスを継承したクラス)です。

用語

用語	意味
コンテナオブジェクト	別のオブジェクトを内部に有するオブジェクトのこと

コンテナオブジェクトの例：

エクセルであれば、エクセルファイルは複数のエクセルシートを持つ。エクセルシートは複数のセルを持つ。

このとき、エクセルファイルはエクセルシートのコンテナオブジェクト。また、エクセルシートはセルのコンテナオブジェクト。

```

import sys
from PySide6.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout, QPushButton, QLabel, QLineEdit

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("PySide6 アーキテクチャ例")
        self.resize(400, 200)

        # 中央ウィジェットとレイアウトを設定
        central_widget = QWidget()
        self.setCentralWidget(central_widget)
        layout = QVBoxLayout(central_widget)

        # UI要素を作成 (最上層の小さな要素たち)
        self.label = QLabel("これは QLabel")
        self.line_edit = QLineEdit("テキストを入力")
        self.button = QPushButton("クリック")

        # シグナルと関数を接続
        self.button.clicked.connect(self.my_function)

        # レイアウトに追加
        layout.addWidget(self.label)
        layout.addWidget(self.line_edit)
        layout.addWidget(self.button)

    def my_function(self):
        print("ボタンがクリックされました")

if __name__ == "__main__":
    # 1. QApplication を作成 (最下層の青い土台)
    app = QApplication()

    # 2. QMainWindow を作成 (オレンジの層)
    window = MainWindow()

    # 3. ウィンドウを表示してイベントループ開始
    window.show()
    app.exec_()

```

「イベント」と「イベント処理の仕組み」

アプリケーション上で生じる様々な出来事のことを「イベント」と言います。

イベントの例:

- クリックした
- クリック状態が終わった
- ドラッグ開始した
- とあるウィジェットにフォーカスが来た
- とあるウィジェットからフォーカスが離れた
- とある QLineEdit の入力内容が変更された

QApplicationは、QWidgetの状態を常時監視しています。たとえば、ウィンドウ上のボタンがクリックされたとします。そのときの処理の流れは以下のようになります。

1. QApplicationは、マウスクリックイベントを検知
2. QApplicationは、マウスクリックイベントが生じた座標を取得
3. QApplicationは、マウスクリックイベントが生じたところに QPushButton があったということを検知
4. QApplicationは、マウスクリックイベントを当該QPushButtonに送信
5. QPushButtonは、QApplicationから送信されたマウスクリックイベントを受信
6. QPushButtonは、あらかじめ関連づけられていた関数やメソッドを実行

基本構造

```

QApplication (アプリケーション全体を管理)
└── QWidget (トップレベル - 独立したウィンドウ)
    ├── QWidget (子ウィジェット)
    │   ├── QPushButton (子ウィジェット)
    │   └── QLabel (子ウィジェット)
    └── QWidget (別のトップレベルウィンドウ)
        └── QDialog (ダイアログウィンドウ)

```

まとめ - 2つの中核クラス

QApplication

- **役割:** アプリケーション全体の管理者
- **責任:** イベントループ、システム設定、全ウィジェットの統括
- **特徴:** アプリケーション内で必ず1つだけ存在
- **管理対象:** すべてのQWidgetインスタンス

QWidget

- **役割:** すべてのUI要素の基底クラス
- **特徴:** 親子関係による階層構造を形成
- **分類:**
- **トップレベルウィジェット:** 親を持たない独立したウィンドウ
- **子ウィジェット:** 他のウィジェット内に配置されるコンポーネント

親子関係の重要性

```
# トップレベル（親なし）
main_window = QMainWindow() # QApplication.topLevelWidgets()に含まれる

# 子ウィジェット（親あり）
button = QPushButton(main_window) # main_windowの子
label = QLabel(main_window) # main_windowの子
```

この階層構造により、Qtは効率的なメモリ管理、イベント処理、ウィジェットのライフサイクル管理を実現しています。

QApplication

QApplicationは、PySide6アプリケーション全体を管理するクラスです。

アプリケーション内でこのクラスのインスタンスは1つしか生成できません。

複数のインスタンスを作成しようとすると例外が発生します。

インポート

```
from PySide6.QtWidgets import QApplication
```

基本的な使用方法

QWidgetについては後述。

以下が最低限のコード。

```
from PySide6.QtWidgets import QApplication, QWidget # 必要なPySide6のクラスをインポート

app = QApplication() # QApplicationインスタンスを作成（アプリケーション全体を管理）
window = QWidget() # 基本的なウィンドウを作成
window.show() # ウィンドウを表示
app.exec() #
```

sys.exit() の引数とすることで、異常終了を検知することができる。

```
from PySide6.QtWidgets import QApplication, QWidget

app = QApplication()
window = QWidget()
window.show()
sys.exit(app.exec()) # イベントループを開始し、アプリケーションの終了コードで終了
```

引数等を受け取って利用することも考えられる。

以下は、コマンドライン引数を受け取って利用する例。

```
import sys # システムモジュールをインポート（コマンドライン引数の処理に必要）
from PySide6.QtWidgets import QApplication, QWidget

app = QApplication(sys.argv) # sys.argv を受け取って利用する
window = QWidget()
window.show()
sys.exit(app.exec())
```

主要なメソッド

コンストラクタ

```
QApplication(argv)
```

パラメータ: | パラメータ | 説明 ||-----|----| | `argv` | list: コマンドライン引数のリスト（通常は `sys.argv`） |

戻り値: | 戻り値 | 説明 ||-----|----| | QApplication | アプリケーションのインスタンス |

例外: | 例外 | 説明 ||-----|----| | RuntimeError | 既に QApplication インスタンスが存在する場合 |

exec()

```
app.exec()
```

アプリケーションのメインイベントループを開始します。ユーザーがアプリケーションを終了するまでブロックされます。

パラメータ: | パラメータ | 説明 ||-----|----| | なし | - |

戻り値: | 戻り値 | 説明 ||-----|----| | int | アプリケーションの終了コード |

quit()

```
app.quit()
```

アプリケーションを終了します。イベントループを終了し、`exec()` から戻ります。

パラメータ: | パラメータ | 説明 ||-----|----| | なし | - |

戻り値: | 戻り値 | 説明 ||-----|----| | None | - |

instance()

```
QApplication.instance()
```

現在の QApplication インスタンスを取得します。アプリケーション内で1つだけ存在するインスタンスへのアクセスに使用されます。

パラメータ: | パラメータ | 説明 ||-----|----| | なし | - |

戻り値: | 戻り値 | 説明 ||-----|----| | QApplication | 現在のアプリケーションインスタンス。インスタンスが存在しない場合はNone |

インスタンス変数とプロパティ

QApplication クラスは、アプリケーション全体の状態を管理するための様々なプロパティを提供しています。

初心者にとって重要なプロパティ

プロパティ	説明	設定メソッド	取得メソッド
style	アプリケーションのスタイル	setStyle(style)	style()
font	アプリケーションのデフォルトフォント	setFont(font)	font()
applicationName	アプリケーションの名前	setApplicationName(name)	applicationName()
applicationVersion	アプリケーションのバージョン	setApplicationVersion(version)	applicationVersion()

そのほかのプロパティの例

プロパティ	説明	設定メソッド	取得メソッド
organizationName	組織名	setOrganizationName(name)	organizationName()
organizationDomain	組織のドメイン名	setOrganizationDomain(domain)	organizationDomain()
palette	アプリケーションのパレット	setPalette(palette)	palette()
layoutDirection	レイアウトの方向	setLayoutDirection(direction)	layoutDirection()

状態プロパティ（初心者向け）

プロパティ	説明	取得メソッド
activeWindow	現在アクティブなウィンドウ	activeWindow()
focusWidget	現在フォーカスされているウィジエット	focusWidget()
mouseButtons	現在押されているマウスボタン	mouseButtons()
keyboardModifiers	現在押されているキーボード修飾子	keyboardModifiers()

使用例

```
app = QApplication(sys.argv)

# アプリケーション情報の設定
app.setApplicationName("My App")
app.setApplicationVersion("1.0.0")
app.setOrganizationName("My Company")
app.setOrganizationDomain("example.com")

# スタイルとフォントの設定
app.setStyle("Fusion") # モダンなスタイル
app.setFont(QFont("Arial", 10))

# 状態の取得
active_window = app.activeWindow()
focused_widget = app.focusWidget()
```

QWidgetのイベントハンドラ

QWidgetクラスは、様々なイベントを処理するためのハンドラメソッドを提供しています。こ

これらのメソッドは、必要に応じてオーバーライドして使用できます。

イベントハンドラー一覧

カテゴリ	メソッド	説明
ウィンドウ	closeEvent(self, event)	ウィンドウを閉じる時
	showEvent(self, event)	ウィンドウが表示される時
	hideEvent(self, event)	ウィンドウが隠される時
	moveEvent(self, event)	ウィンドウが移動される時
	resizeEvent(self, event)	ウィンドウのサイズが変更される時
マウス	mousePressEvent(self, event)	マウスボタンが押された時
	mouseReleaseEvent(self, event)	マウスボタンが離された時
	mouseMoveEvent(self, event)	マウスが移動した時
	mouseDoubleClickEvent(self, event)	マウスがダブルクリックされた時
キーボード	wheelEvent(self, event)	マウスホイールが回転した時
	keyPressEvent(self, event)	キーが押された時
	keyReleaseEvent(self, event)	キーが離された時
フォーカス	focusInEvent(self, event)	ウィジェットがフォーカスを得た時
	focusOutEvent(self, event)	ウィジェットがフォーカスを失った時
ドラッグ & ドロップ	dragEnterEvent(self, event)	ドラッグがウィジェットに入った時
	dragLeaveEvent(self, event)	ドラッグがウィジェットから出た時
	dropEvent(self, event)	ドロップされた時
描画	paintEvent(self, event)	ウィジェットの描画が必要な時

イベントハンドラの使用例

ウィンドウを閉じる時の処理

```
class MyWindow(QWidget):
    def closeEvent(self, event):
        print("ウィンドウを閉じます")
        event.accept() # イベントを処理したことを通知
```

マウスの動きを追跡

```
class MyWindow(QWidget):
    def mouseMoveEvent(self, event):
        print(f"マウス位置: ({event.x()}, {event.y()})")
```

キー入力を処理

```
class MyWindow(QWidget):
    def keyPressEvent(self, event):
        if event.key() == Qt.Key_Escape:
            print("ESCキーが押されました")
            self.close()
```

使用例

基本的なアプリケーション

```
import sys
```

```
from PySide6.QtWidgets import QApplication, QMainWindow

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My App")

    if __name__ == "__main__":
        app = QApplication(sys.argv)
        window = MainWindow()
        window.show()
        sys.exit(app.exec())
```

注意事項

- 1つのアプリケーションにつき、QApplicationのインスタンスは1つだけ作成してください
- `sys.exit(app.exec())` でアプリケーションを適切に終了させることが重要です

QWidget

QWidgetは、PySide6のすべてのUIコンポーネントの基底クラスです。
独立したウィンドウまたは他のウィジェットの子として使用できます。

インポート

```
from PySide6.QtWidgets import QWidget
```

基本的な使用方法

```
from PySide6.QtWidgets import QWidget, QVBoxLayout

widget = QWidget()
layout = QVBoxLayout()
widget.setLayout(layout)
```

主要なメソッド

レイアウト管理

setLayout(layout)

```
layout = QVBoxLayout()
widget.setLayout(layout)
```

ウィジェットにレイアウトを設定します。

パラメータ: - `layout` (QLayout): 設定するレイアウト

layout()

```
current_layout = widget.layout()
```

現在のレイアウトを取得します。

戻り値: QLayout - 現在のレイアウト (設定されていない場合はNone)

サイズ設定

resize(width, height)

```
widget.resize(400, 300)
```

ウィジェットのサイズを設定します。

パラメータ: - `width` (int): 幅 - `height` (int): 高さ

setFixedSize(width, height)

```
widget.setFixedSize(400, 300)
```

ウィジェットの固定サイズを設定し、リサイズを無効にします。

パラメータ:- `width` (int): 固定幅 - `height` (int): 固定高さ

表示制御

show()

```
widget.show()
```

ウィジェットを表示します。

hide()

```
widget.hide()
```

ウィジェットを非表示にします。

setVisible(visible)

```
widget.setVisible(True) # 表示  
widget.setVisible(False) # 非表示
```

ウィジェットの表示/非表示を設定します。

パラメータ:- `visible` (bool): True で表示、False で非表示

有効/無効制御

setEnabled(enabled)

```
widget.setEnabled(False) # 無効化  
widget.setEnabled(True) # 有効化
```

ウィジェットの有効/無効を設定します。

パラメータ:- `enabled` (bool): True で有効、False で無効

isEnabled()

```
is_enabled = widget.isEnabled()
```

ウィジェットが有効かどうかを確認します。

戻り値: bool - 有効な場合True

使用例

基本的なウィジェット

```
import sys
from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QPushButton

class MyWidget(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("My Widget")
        self.resize(300, 200)

        # レイアウトの設定
        layout = QVBoxLayout()
        self.setLayout(layout)

        # 子ウィジェットの追加
        label = QLabel("Hello World!")
        button = QPushButton("Click Me")

        layout.addWidget(label)
        layout.addWidget(button)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    widget = MyWidget()
    widget.show()
    sys.exit(app.exec())
```

コンテナウィジェット

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QLabel

# 他のウィジェットのコンテナとして使用
container = QWidget()
layout = QVBoxLayout()
container.setLayout(layout)

for i in range(3):
    label = QLabel(f"Label {i+1}")
    layout.addWidget(label)
```

注意事項

- QWidgetは他のすべてのウィジェットクラスの基底クラスです
- レイアウトを設定する前に子ウィジェットを追加しないでください
- 親を持たないQWidgetは独立したウィンドウとして表示されます

QMainWindow

QMainWindowは、アプリケーションのメインウィンドウを作成するためのクラスです。ツールバー、メニューバー、ステータスバー、中央ウィジェットなどの標準的なウィンドウ要素を提供します。

インポート

```
from PySide6.QtWidgets import QMainWindow
```

基本的な使用方法

```
from PySide6.QtWidgets import QMainWindow, QWidget

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("マイアプリ")
        self.setGeometry(100, 100, 800, 600)
```

主要なメソッド

ウィンドウ設定

setTitle(title)

```
self.setWindowTitle("ウィンドウタイトル")
```

ウィンドウのタイトルを設定します。

パラメータ:- `title` (str): ウィンドウのタイトル

setGeometry(x, y, width, height)

```
self.setGeometry(100, 100, 800, 600)
```

ウィンドウの位置とサイズを設定します。

パラメータ:- `x` (int): ウィンドウの左上角のX座標 - `y` (int): ウィンドウの左上角のY座標 - `width` (int): ウィンドウの幅 - `height` (int): ウィンドウの高さ

中央ウィジェット

setCentralWidget(widget)

```
central_widget = QWidget()
self.setCentralWidget(central_widget)
```

メインウィンドウの中央領域にウィジェットを設定します。

パラメータ: - `widget` (QWidget): 中央に配置するウィジェット

centralWidget()

```
widget = self.centralWidget()
```

現在の中央ウィジェットを取得します。

戻り値: QWidget - 中央ウィジェット

表示制御

show()

```
self.show()
```

ウィンドウを表示します。

hide()

```
self.hide()
```

ウィンドウを非表示にします。

close()

```
self.close()
```

ウィンドウを閉じます。

使用例

基本的なメインウィンドウ

```
import sys
from PySide6.QtWidgets import QApplication, QMainWindow, QLabel, QWidget, QVBoxLayout

class MainWindow(QMainWindow):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("サンプルアプリケーション")
        self.setGeometry(100, 100, 400, 300)

        # 中央ウィジェットの設定
        central_widget = QWidget()
        self.setCentralWidget(central_widget)

        # レイアウトの設定
        layout = QVBoxLayout()
        central_widget.setLayout(layout)

        # ラベルの追加
        label = QLabel("Hello, PySide6!")
        layout.addWidget(label)

if __name__ == "__main__":
    app = QApplication(sys.argv)
```

```
window = MainWindow()
window.show()
sys.exit(app.exec())
```

注意事項

- QMainWindowには必ず中央ウィジェットを設定してください
- レイアウトは中央ウィジェットに対して設定します
- QMainWindow自体には直接レイアウトを設定できません

QLabel

QLabelは、テキストや画像を表示するためのウィジェットです。静的なコンテンツの表示に使用されます。

インポート

```
from PySide6.QtWidgets import QLabel
```

基本的な使用方法

```
from PySide6.QtWidgets import QLabel  
  
label = QLabel("Hello, World!")
```

主要なメソッド

テキスト設定

setText(text)

```
label.setText("新しいテキスト")
```

ラベルに表示するテキストを設定します。

パラメータ:- `text` (str): 表示するテキスト

text()

```
current_text = label.text()
```

現在のテキストを取得します。

戻り値: str - 現在表示されているテキスト

テキスト配置

setAlignment(alignment)

```
from PySide6.QtCore import Qt  
label.setAlignment(Qt.AlignmentFlag.AlignCenter)
```

テキストの配置を設定します。

パラメータ:- `alignment` (Qt.AlignmentFlag): 配置方法 - `Qt.AlignmentFlag.AlignLeft` - 左揃え - `Qt.AlignmentFlag.AlignRight` - 右揃え - `Qt.AlignmentFlag.AlignCenter` - 中央揃え - `Qt.AlignmentFlag.AlignTop` - 上揃え - `Qt.AlignmentFlag.AlignBottom` - 下揃え

スタイル設定

setStyleSheet(styleSheet)

```
label.setStyleSheet("background-color: #ff0000; color: #ffffff;")
```

CSSライクなスタイルシートを設定します。

パラメータ:- `styleSheet` (str): スタイルシート文字列

setFont(font)

```
from PySide6.QtGui import QFont
font = QFont("Arial", 14)
label.setFont(font)
```

フォントを設定します。

パラメータ:- `font` (QFont): 設定するフォント

画像表示

setPixmap(pixmap)

```
from PySide6.QtGui import QPixmap
pixmap = QPixmap("image.png")
label.setPixmap(pixmap)
```

ラベルに画像を表示します。

パラメータ:- `pixmap` (QPixmap): 表示する画像

その他の設定

setWordWrap(on)

```
label.setWordWrap(True)
```

長いテキストの自動改行を有効/無効にします。

パラメータ:- `on` (bool): True で自動改行を有効

使用例

基本的なラベル

```
import sys
from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel
from PySide6.QtCore import Qt

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
```

```

    self.setLayout(layout)

    # 基本的なラベル
    label1 = QLabel("こんにちは！")
    layout.addWidget(label1)

    # 中央揃えのラベル
    label2 = QLabel("中央揃えテキスト")
    label2.setAlignment(Qt.AlignmentFlag.AlignCenter)
    layout.addWidget(label2)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())

```

スタイル付きラベル

```

from PySide6.QtWidgets import QLabel
from PySide6.QtGui import QFont
from PySide6.QtCore import Qt

label = QLabel("スタイル付きラベル")
label.setAlignment(Qt.AlignmentFlag.AlignCenter)
label.setStyleSheet("""
    background-color: #2c3e50;
    color: #ecf0f1;
    border: 2px solid #3498db;
    border-radius: 10px;
    padding: 10px;
""")
label.setFont(QFont("Arial", 16, QFont.Weight.Bold))

```

動的テキスト更新

```

from datetime import datetime
from PySide6.QtWidgets import QLabel, QPushButton, QVBoxLayout, QWidget

class TimeDisplay(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        self.time_label = QLabel("時刻が表示されます")
        self.time_label.setAlignment(Qt.AlignmentFlag.AlignCenter)

        update_button = QPushButton("時刻更新")
        update_button.clicked.connect(self.update_time)

        layout.addWidget(self.time_label)
        layout.addWidget(update_button)

    def update_time(self):
        current_time = datetime.now().strftime("%Y/%m/%d %H:%M:%S")
        self.time_label.setText(current_time)

```

よく使用されるスタイルシートプロパティ

```

/* 背景色 */
background-color: #3498db;

/* 文字色 */
color: #ffffff;

```

```
/* ボーダー */
border: 2px solid #2c3e50;
border-radius: 5px;

/* パディング */
padding: 10px;

/* マージン */
margin: 5px;

/* フォントサイズ */
font-size: 16px;
font-weight: bold;
```

注意事項

- テキストと画像の両方を同時に表示することはできません
- 長いテキストを表示する場合は `setWordWrap(True)` を使用してください
- アライメントは複数の値を組み合わせることができます (例:
`Qt.AlignmentFlag.AlignCenter | Qt.AlignmentFlag.AlignTop`)

QPushButton

QPushButtonは、ユーザーがクリックできるボタンウィジェットです。シグナルとスロットメカニズムを使用してイベント処理を行います。

インポート

```
from PySide6.QtWidgets import QPushButton
```

基本的な使用方法

```
from PySide6.QtWidgets import QPushButton

button = QPushButton("クリックしてください")
button.clicked.connect(some_function)
```

主要なメソッド

テキスト設定

setText(text)

```
button.setText("新しいボタン")
```

ボタンに表示するテキストを設定します。

パラメータ:- `text` (str): ボタンに表示するテキスト

text()

```
current_text = button.text()
```

現在のボタンテキストを取得します。

戻り値: str - 現在表示されているテキスト

イベント処理

clicked.connect(slot)

```
def on_button_click():
    print("ボタンがクリックされました")

button.clicked.connect(on_button_click)
```

ボタンクリック時に呼び出される関数を接続します。

パラメータ:- `slot` (callable): クリック時に呼び出される関数

clicked.disconnect()

```
button.clicked.disconnect()
```

すべてのクリックイベント接続を解除します。

サイズとレイアウト

setFixedSize(width, height)

```
button.setFixedSize(100, 30)
```

ボタンの固定サイズを設定します。

パラメータ:- `width` (int): 固定幅 - `height` (int): 固定高さ

setSizePolicy(policy)

```
from PySide6.QtWidgets import QSizePolicy  
button.setSizePolicy(QSizePolicy.Policy.Expanding, QSizePolicy.Policy.Fixed)
```

ボタンのサイズポリシーを設定します。

スタイル設定

setStyleSheet(styleSheet)

```
button.setStyleSheet("""  
    QPushButton {  
        background-color: #3498db;  
        color: white;  
        border: none;  
        padding: 10px;  
        border-radius: 5px;  
    }  
    QPushButton:hover {  
        background-color: #2980b9;  
    }  
""")
```

CSSライクなスタイルシートを設定します。

パラメータ:- `styleSheet` (str): スタイルシート文字列

setFont(font)

```
from PySide6.QtGui import QFont  
font = QFont("Arial", 12, QFont.Weight.Bold)  
button.setFont(font)
```

ボタンのフォントを設定します。

パラメータ:- `font` (QFont): 設定するフォント

状態制御

setEnabled(enabled)

```
button.setEnabled(False) # ボタンを無効化  
button.setEnabled(True) # ボタンを有効化
```

ボタンの有効/無効を設定します。

パラメータ: - `enabled` (bool): True で有効、False で無効

setCheckable(checkable)

```
button.setCheckable(True)
```

ボタンをトグルボタンとして使用できるようにします。

パラメータ: - `checkable` (bool): True でチェック可能

isChecked()

```
is_pressed = button.isChecked()
```

チェック可能なボタンの状態を取得します。

戻り値: bool - チェックされている場合True

使用例

基本的なボタン

```
import sys
from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QPushButton, QLabel

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        self.label = QLabel("ボタンをクリックしてください")
        layout.addWidget(self.label)

        button = QPushButton("クリック")
        button.clicked.connect(self.on_button_click)
        layout.addWidget(button)

    def on_button_click(self):
        self.label.setText("ボタンがクリックされました!")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

複数のボタン

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QPushButton, QLabel

class MultiButtonWidget(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        self.status_label = QLabel("何も押されていません")
        layout.addWidget(self.status_label)

        # 複数のボタンを作成
        buttons = ["ボタン1", "ボタン2", "ボタン3"]
        for button_text in buttons:
```

```

button = QPushButton(button_text)
button.clicked.connect(lambda checked, text=button_text: self.on_button_click(text))
layout.addWidget(button)

def on_button_click(self, button_name):
    self.status_label.setText(f"{button_name} がクリックされました")

```

スタイル付きボタン

```

from PySide6.QtWidgets import QPushButton
from PySide6.QtGui import QFont

def create_styled_button(text, color="#3498db"):
    button = QPushButton(text)
    button.setStyleSheet(f"""
        QPushButton {{
            background-color: {color};
            color: white;
            border: none;
            padding: 12px 24px;
            border-radius: 6px;
            font-size: 14px;
            font-weight: bold;
        }}
        QPushButton:hover {{
            background-color: #2980b9;
        }}
        QPushButton:pressed {{
            background-color: #21618c;
        }}
        QPushButton:disabled {{
            background-color: #bdc3c7;
            color: #7f8c8d;
        }}
    """)
    return button

# 使用例
save_button = create_styled_button("保存", "#27ae60")
delete_button = create_styled_button("削除", "#e74c3c")
cancel_button = create_styled_button("キャンセル", "#95a5a6")

```

トグルボタン

```

from PySide6.QtWidgets import QPushButton, QLabel, QVBoxLayout, QWidget

class ToggleButtonExample(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        self.status_label = QLabel("オフ")
        layout.addWidget(self.status_label)

        self.toggle_button = QPushButton("ON/OFF")
        self.toggle_button.setCheckable(True)
        self.toggle_button.clicked.connect(self.on_toggle)
        layout.addWidget(self.toggle_button)

    def on_toggle(self):
        if self.toggle_button.isChecked():
            self.status_label.setText("オン")
            self.toggle_button.setText("オフにする")
        else:
            self.status_label.setText("オフ")
            self.toggle_button.setText("オンにする")

```

よく使用されるスタイルシートプロパティ

```
/* 基本スタイル */
QPushButton {
    background-color: #3498db;
    color: white;
    border: 2px solid #2980b9;
    padding: 8px 16px;
    border-radius: 4px;
}

/* ホバー状態 */
QPushButton:hover {
    background-color: #2980b9;
}

/* 押下状態 */
QPushButton:pressed {
    background-color: #21618c;
}

/* 無効状態 */
QPushButton:disabled {
    background-color: #bdcb3c7;
    color: #7f8c8d;
}

/* チェック状態 (トグルボタン) */
QPushButton:checked {
    background-color: #27ae60;
}
```

注意事項

- clicked.connect() で関数を接続する際、引数が必要な場合は lambda を使用してください
- スタイルシートでは疑似状態 (:hover, :pressed など) を活用して見た目を向上させることができます
- トグルボタンを使用する場合は setCheckable(True) を忘れずに設定してください

QLineEdit クラス

概要

`QLineEdit` は、1行のテキスト入力を可能にするウィジェットです。ユーザーがテキストを入力・編集するための基本的なコンポーネントで、フォーム、検索ボックス、設定画面などで広く使用されます。

基本的な使用方法

```
from PySide6.QtWidgets import QApplication, QLineEdit, QVBoxLayout, QWidget
import sys

app = QApplication(sys.argv)
window = QWidget()
layout = QVBoxLayout()

# 基本的なQLineEditの作成
line_edit = QLineEdit()
line_edit.setPlaceholderText("ここにテキストを入力")
layout.addWidget(line_edit)

window.setLayout(layout)
window.show()
sys.exit(app.exec())
```

主要なプロパティとメソッド

テキスト操作

メソッド	説明	例
<code>setText(text)</code>	テキストを設定	<code>line_edit.setText("Hello")</code>
<code>text()</code>	現在のテキストを取得	<code>current_text = line_edit.text()</code>
<code>clear()</code>	テキストをクリア	<code>line_edit.clear()</code>
<code>insert(text)</code>	カーソル位置にテキストを挿入	<code>line_edit.insert("World")</code>

プレースホルダーテキスト

メソッド	説明	例
<code>setPlaceholderText(text)</code>	プレースホルダーテキストを設定	<code>line_edit.setPlaceholderText("名前を入力")</code>
<code>placeholderText()</code>	プレースホルダーテキストを取得	<code>placeholder = line_edit.placeholderText()</code>

入力制限と検証

メソッド	説明	例
<code>setMaxLength(length)</code>	最大文字数を設定	<code>line_edit.setMaxLength(50)</code>
<code>maxLength()</code>	最大文字数を取得	<code>max_len = line_edit.maxLength()</code>
<code>setValidator(validator)</code>	入力バリデーターを設定	<code>line_edit.setValidator(QIntValidator())</code>
<code>setInputMask(mask)</code>	入力マスクを設定	<code>line_edit.setInputMask("000.000.000.000")</code>

表示モード

メソッド	説明	例
<code>setEchoMode(mode)</code>	文字の表示モードを設定	<code>line_edit.setEchoMode(QLineEdit.Password)</code>
<code>echoMode()</code>	現在の表示モードを取得	<code>mode = line_edit.echoMode()</code>
<code>setReadOnly(readonly)</code>	読み取り専用モードの設定	<code>line_edit.setReadOnly(True)</code>
<code>isReadOnly()</code>	読み取り専用かどうかを確認	<code>is_READONLY = line_edit.isReadOnly()</code>

カーソルとテキスト選択

メソッド	説明	例
<code>setCursorPosition(pos)</code>	カーソル位置を設定	<code>line_edit.setCursorPosition(5)</code>
<code>cursorPosition()</code>	現在のカーソル位置を取得	<code>pos = line_edit.cursorPosition()</code>
<code>selectAll()</code>	すべてのテキストを選択	<code>line_edit.selectAll()</code>
<code>setSelection(start, length)</code>	指定範囲のテキストを選択	<code>line_edit.setSelection(0, 5)</code>
<code>selectedText()</code>	選択されたテキストを取得	<code>selected = line_edit.selectedText()</code>

エコーモード

QLineEditは、以下のエコーモードをサポートします：

モード	説明	用途
<code>Normal</code>	通常表示（デフォルト）	一般的なテキスト入力
<code>NoEcho</code>	文字を表示しない	機密情報の入力
<code>Password</code>	アスタリスクで表示	パスワード入力
<code>PasswordEchoOnEdit</code>	編集時のみ文字を表示	ユーザビリティを考慮したパスワード入力

主要なシグナル

シグナル	説明	使用例
<code>textChanged(text)</code>	テキストが変更された時	<code>line_edit.textChanged.connect(on_text_changed)</code>
<code>textEdited(text)</code>	ユーザーがテキストを編集した時	<code>line_edit.textEdited.connect(on_text_edited)</code>
<code>returnPressed()</code>	Enterキーが押された時	<code>line_edit.returnPressed.connect(on_enter_pressed)</code>
<code>editingFinished()</code>	編集が完了し	<code>line_edit.editingFinished.connect(on_edit_finished)</code>

シグナル	説明	使用例
	た時	
selectionChanged()	テキスト選択が変更された時	line_edit.selectionChanged.connect(on_selection_changed)

実用的な使用例

1. パスワード入力フィールド

```
password_edit = QLineEdit()
password_edit.setEchoMode(QLineEdit.Password)
password_edit.setPlaceholderText("パスワードを入力")
```

2. 数値のみの入力

```
from PySide6.QtGui import QIntValidator

number_edit = QLineEdit()
number_edit.setValidator(QIntValidator(0, 100)) # 0-100の整数のみ
number_edit.setPlaceholderText("0-100の数値を入力")
```

3. IPアドレス入力

```
ip_edit = QLineEdit()
ip_edit.setInputMask("000.000.000.000;_")
ip_edit.setPlaceholderText("192.168.1.1")
```

4. リアルタイム検索

```
search_edit = QLineEdit()
search_edit.setPlaceholderText("検索...")
search_edit.textChanged.connect(perform_search)

def perform_search(text):
    # 検索処理を実行
    print(f"検索中: {text}")
```

ベストプラクティス

1. 適切なプレースホルダーの使用

```
# 良い例
line_edit.setPlaceholderText("例: user@example.com")

# 避けるべき例
line_edit.setPlaceholderText("テキストを入力")
```

2. 入力検証の実装

```
# 入力後の検証
def validate_email(text):
    if "@" not in text:
        line_edit.setStyleSheet("border: 2px solid red")
    else:
        line_edit.setStyleSheet("border: 2px solid green")
```

```
line_edit.textChanged.connect(validate_email)
```

3. 適切なサイズ設定

```
# 固定幅の設定  
line_edit.setFixedWidth(200)  
  
# 最小・最大幅の設定  
line_edit.setMinimumWidth(100)  
line_edit.setMaximumWidth(300)
```

注意事項

1. **パフォーマンス**: `textChanged` シグナルは文字入力のたびに発火するため、重い処理は避ける
2. **バリデーション**: ユーザー体験を考慮して、リアルタイム検証と最終検証を適切に使い分ける
3. **アクセシビリティ**: 視覚的な手がかりだけでなく、プレースホルダーテキストで機能を明確に説明する

関連するクラス

- **QTextEdit**: 複数行のテキスト編集
- **QValidator**: 入力検証用のベースクラス
- **QCompleter**: オートコンプリート機能
- **QLabel**: ラベル表示（QLineEditと組み合わせてフォーム作成）

参考リンク

- [Qt公式ドキュメント - QLineEdit](#)
- [PySide6公式ドキュメント](#)

QCheckBox クラス

概要

`QCheckBox` は、ユーザーがオプションを選択（チェック）または選択解除（アンチェック）できるウィジェットです。複数の選択肢から複数のアイテムを選択する際に使用され、フォーム、設定画面、オプション選択などで広く活用されます。

基本的な使用方法

```
from PySide6.QtWidgets import QApplication, QCheckBox, QVBoxLayout, QWidget
import sys

app = QApplication(sys.argv)
window = QWidget()
layout = QVBoxLayout()

# 基本的なQCheckBoxの作成
checkbox = QCheckBox("同意する")
layout.addWidget(checkbox)

window.setLayout(layout)
window.show()
sys.exit(app.exec())
```

主要なプロパティとメソッド

チェック状態の操作

メソッド	説明	例
<code>setChecked(bool checked)</code>	チェック状態を設定	<code>checkbox.setChecked(True)</code>
<code>isChecked()</code>	チェック状態を取得	<code>is_checked = checkbox.isChecked()</code>
<code>toggle()</code>	チェック状態を切り替え	<code>checkbox.toggle()</code>
<code>setCheckState(Qt.CheckState state)</code>	詳細なチェック状態を設定	<code>checkbox.setCheckState(Qt.PartiallyChecked)</code>
<code>checkState()</code>	詳細なチェック状態を取得	<code>state = checkbox.checkState()</code>

テキストとアイコン

メソッド	説明	例
<code>setText(str text)</code>	ラベルテキストを設定	<code>checkbox.setText("新しいオプション")</code>
<code>text()</code>	ラベルテキストを取得	<code>text = checkbox.text()</code>
<code>setIcon(QIcon icon)</code>	アイコンを設定	<code>checkbox.setIcon(QIcon("icon.png"))</code>
<code>icon()</code>	アイコンを取得	<code>icon = checkbox.icon()</code>

三状態チェックボックス

メソッド	説明	例
<code>setTristate(bool tristate)</code>	三状態モードを有効/無効	<code>checkbox.setTristate(True)</code>
<code>isTristate()</code>	三状態モードかどうかを確認	<code>is_tri = checkbox.isTristate()</code>

チェック状態

QCheckBoxは以下の状態を持ちます：

状態	説明	値
Qt.Unchecked	チェックなし	0
Qt.PartiallyChecked	部分的にチェック（三状態モード）	1
Qt.Checked	チェック済み	2

主要なシグナル

シグナル	説明	使用例
toggled(checked)	チェック状態が変更された時	checkbox.toggled.connect(on_toggled)
stateChanged(state)	状態が変更された時（三状態対応）	checkbox.stateChanged.connect(on_state_changed)
clicked(checked)	クリックされた時	checkbox.clicked.connect(on_clicked)

実用的な使用例

1. 基本的なオプション選択

```
# 複数のオプション
options = ["メール通知", "SMS通知", "プッシュ通知"]
checkboxes = []

for option in options:
    checkbox = QCheckBox(option)
    checkboxes.append(checkbox)
    layout.addWidget(checkbox)
```

2. 利用規約同意チェック

```
terms_checkbox = QCheckBox("利用規約に同意する")
terms_checkbox.toggled.connect(lambda checked: submit_button.setEnabled(checked))
```

3. 三状態チェックボックス（親子関係）

```
parent_checkbox = QCheckBox("すべて選択")
parent_checkbox.setTristate(True)

child_checkboxes = [
    QCheckBox("オプション1"),
    QCheckBox("オプション2"),
    QCheckBox("オプション3")
]

def update_parent_state():
    checked_count = sum(1 for cb in child_checkboxes if cb.isChecked())
    if checked_count == 0:
        parent_checkbox.setCheckState(Qt.Unchecked)
    elif checked_count == len(child_checkboxes):
        parent_checkbox.setCheckState(Qt.Checked)
    else:
        parent_checkbox.setCheckState(Qt.PartiallyChecked)
```

```

for checkbox in child_checkboxes:
    checkbox.toggled.connect(update_parent_state)

```

4. 条件付きオプション

```

enable_feature = QCheckBox("高度な機能を有効にする")
advanced_options = QCheckBox("詳細オプション")

# 依存関係の設定
enable_feature.toggled.connect(advanced_options.setEnabled)
advanced_options.setEnabled(False) # 初期状態では無効

```

QRadioButton との比較

QCheckBoxとよく似たウィジェットにQRadioButtonがあります：

特徴	QCheckBox	QRadioButton
選択方式	複数選択可能	単一選択（グループ内）
用途	オプション設定、機能有効/無効	択一選択
表示	四角いチェックマーク	丸いラジオボタン
状態	三状態サポート	二状態のみ

QRadioButton の基本的な使用方法

```

from PySide6.QtWidgets import QRadioButton, QButtonGroup

# ラジオボタンの作成
radio1 = QRadioButton("オプション1")
radio2 = QRadioButton("オプション2")
radio3 = QRadioButton("オプション3")

# グループ化（排他制御）
button_group = QButtonGroup()
button_group.addButton(radio1)
button_group.addButton(radio2)
button_group.addButton(radio3)

# デフォルト選択
radio1.setChecked(True)

```

スタイリングとカスタマイズ

CSS スタイルシートの例

```

checkbox.setStyleSheet("""
    QCheckBox {
        font-size: 14px;
        color: #333;
        spacing: 10px;
    }
    QCheckBox::indicator {
        width: 18px;
        height: 18px;
    }
    QCheckBox::indicator:unchecked {
        border: 2px solid #cccccc;
        background-color: white;
        border-radius: 3px;
    }
    QCheckBox::indicator:checked {
        border: 2px solid #007acc;
    }
""")

```

```

        background-color: #007acc;
        border-radius: 3px;
    }
    QCheckBox::indicator:checked {
        image: url(checkmark.png);
    }
    """)

```

ベストプラクティス

1. 明確なラベルの使用

```

# 良い例
checkbox = QCheckBox("毎日メール通知を受け取る")

# 避けるべき例
checkbox = QCheckBox("メール")

```

2. 適切なグループ化

```

# 関連するオプションをグループ化
notification_group = QGroupBox("通知設定")
notification_layout = QVBoxLayout()

email_cb = QCheckBox("メール通知")
sms_cb = QCheckBox("SMS通知")
push_cb = QCheckBox("プッシュ通知")

notification_layout.addWidget(email_cb)
notification_layout.addWidget(sms_cb)
notification_layout.addWidget(push_cb)
notification_group.setLayout(notification_layout)

```

3. 状態変更の処理

```

def handle_option_changed(checked):
    if checked:
        print("オプションが有効になりました")
        # 関連する機能を有効化
    else:
        print("オプションが無効になりました")
        # 関連する機能を無効化

checkbox.toggled.connect(handle_option_changed)

```

注意事項

1. アクセシビリティ: ラベルは簡潔で分かりやすくする
2. ユーザビリティ: 関連するオプションは視覚的にグループ化する
3. 状態管理: 複雑な依存関係がある場合は、状態管理を適切に設計する
4. パフォーマンス: 大量のチェックボックスがある場合は、レイアウトの最適化を検討する

関連するクラス

- **QRadioButton**: 単一選択用のラジオボタン
- **QButtonGroup**: ボタンのグループ管理
- **QGroupBox**: ウィジェットのグループ化コンテナ
- **QAbstractButton**: ボタン系ウィジェットの基底クラス

参考リンク

- [Qt公式ドキュメント - QCheckBox](#)
- [Qt公式ドキュメント - QRadioButton](#)
- [PySide6公式ドキュメント](#)

QComboBox クラス

概要

`QComboBox` は、ユーザーが複数の選択肢から一つを選択できるドロップダウンリストウィジェットです。省スペースで多くの選択肢を提供でき、フォーム、設定画面、フィルタリング機能などで広く使用されます。編集可能・編集不可の両方のモードをサポートします。

基本的な使用方法

```
from PySide6.QtWidgets import QApplication, QComboBox, QVBoxLayout, QWidget
import sys

app = QApplication(sys.argv)
window = QWidget()
layout = QVBoxLayout()

# 基本的なQComboBoxの作成
combo = QComboBox()
combo.addItems(["選択肢1", "選択肢2", "選択肢3"])
layout.addWidget(combo)

window.setLayout(layout)
window.show()
sys.exit(app.exec())
```

主要なプロパティとメソッド

アイテムの追加・削除

メソッド	説明	例
<code>addItem(text, userData)</code>	アイテムを追加	<code>combo.addItem("新しい項目")</code>
<code>addItems(texts)</code>	複数のアイテムを追加	<code>combo.addItems(["項目1", "項目2"])</code>
<code>insertItem(index, text)</code>	指定位置にアイテムを挿入	<code>combo.insertItem(1, "挿入項目")</code>
<code>removeItem(index)</code>	指定インデックスのアイテムを削除	<code>combo.removeItem(2)</code>
<code>clear()</code>	すべてのアイテムを削除	<code>combo.clear()</code>

選択状態の操作

メソッド	説明	例
<code>setCurrentIndex(index)</code>	インデックスで選択を設定	<code>combo.setCurrentIndex(1)</code>
<code>setCurrentText(text)</code>	テキストで選択を設定	<code>combo.setCurrentText("項目2")</code>
<code>currentIndex()</code>	現在の選択インデックスを取得	<code>index = combo.currentIndex()</code>
<code>currentText()</code>	現在の選択テキストを取得	<code>text = combo.currentText()</code>
<code>currentData()</code>	現在の選択データを取得	<code>data = combo.currentData()</code>

アイテム情報の取得

メソッド	説明	例
<code>count()</code>	アイテム数を取得	<code>total = combo.count()</code>
<code>itemText(index)</code>	指定インデックスのテキストを取得	<code>text = combo.itemText(0)</code>
<code>itemData(index)</code>	指定インデックスのデータを取得	<code>data = combo.itemData(0)</code>
<code>findText(text)</code>	テキストでインデックスを検索	<code>index = combo.findText("項目名")</code>

編集機能

メソッド	説明	例
<code>setEditable(edittable)</code>	編集可能モードを設定	<code>combo.setEditable(True)</code>
<code>isEditable()</code>	編集可能かどうかを確認	<code>editable = combo.isEditable()</code>
<code>setInsertPolicy(policy)</code>	挿入ポリシーを設定	<code>combo.setInsertPolicy(QComboBox.InsertAtTop)</code>
<code>insertPolicy()</code>	現在の挿入ポリシーを取得	<code>policy = combo.insertPolicy()</code>

挿入ポリシー

編集可能モードでのアイテム挿入動作を制御します：

ポリシー	説明
<code>NoInsert</code>	新しいアイテムを挿入しない
<code>InsertAtTop</code>	リストの先頭に挿入
<code>InsertAtCurrent</code>	現在の位置に挿入
<code>InsertAtBottom</code>	リストの末尾に挿入
<code>InsertAfterCurrent</code>	現在の位置の後に挿入
<code>InsertBeforeCurrent</code>	現在の位置の前に挿入
<code>InsertAlphabetically</code>	アルファベット順に挿入

主要なシグナル

シグナル	説明	使用例
<code>currentIndexChanged(index)</code>	選択インデックスが変更された時	<code>combo.currentIndexChanged.connect(on_index_changed)</code>

シグナル	説明	使用例
currentTextChanged(text)	選択テキストが変更された時	combo.currentTextChanged.connect(on_text_changed)
activated(index)	ユーザーがアイテムを選択した時	combo.activated.connect(on_activated)
editTextChanged(text)	編集テキストが変更された時	combo.editTextChanged.connect(on_edit_changed)

実用的な使用例

1. 基本的な選択リスト

```
# 国選択
country_combo = QComboBox()
countries = ["日本", "アメリカ", "イギリス", "フランス", "ドイツ"]
country_combo.addItems(countries)

# 選択変更の処理
def on_country_changed(text):
    print(f"選択された国: {text}")

country_combo.currentTextChanged.connect(on_country_changed)
```

2. データ付きアイテム

```
# 言語選択 (表示名とコードを分離)
language_combo = QComboBox()
```

```

languages = [
    ("日本語", "ja"),
    ("English", "en"),
    ("Français", "fr"),
    ("Deutsch", "de")
]

for display_name, code in languages:
    language_combo.addItem(display_name, code)

# 選択されたコードを取得
def get_selected_language_code():
    return language_combo.currentData()

```

3. 編集可能なコンボボックス

```

# 検索履歴付きの検索ボックス
search_combo = QComboBox()
search_combo.setEditable(True)
search_combo.setInsertPolicy(QComboBox.InsertAtTop)

# 既存の検索履歴
search_history = ["PySide6", "Qt Framework", "Python GUI"]
search_combo.addItems(search_history)

def perform_search():
    query = search_combo.currentText()
    if query and query not in search_history:
        search_combo.insertItem(0, query)
        search_history.insert(0, query)
    print(f"検索: {query}")

search_combo.activated.connect(perform_search)

```

4. 動的なアイテム更新

```

category_combo = QComboBox()
item_combo = QComboBox()

# カテゴリデータ
categories = {
    "果物": ["りんご", "バナナ", "オレンジ"],
    "野菜": ["にんじん", "じゃがいも", "玉ねぎ"],
    "肉類": ["牛肉", "豚肉", "鶏肉"]
}

category_combo.addItems(list(categories.keys()))

def update_items(category):
    item_combo.clear()
    if category in categories:
        item_combo.addItems(categories[category])

category_combo.currentTextChanged.connect(update_items)
# 初期値の設定
update_items(category_combo.currentText())

```

5. カスタムアイテム表示

```

from PySide6.QtGui import QIcon

# アイコン付きコンボボックス
status_combo = QComboBox()

statuses = [
    ("オンライン", "online.png"),

```

```

        ("取り込み中", "busy.png"),
        ("離席中", "away.png"),
        ("オフライン", "offline.png")
    ]

    for text, icon_path in statuses:
        status_combo.addItem(QIcon(icon_path), text)

```

フィルタリング機能の実装

コンプリーター付きコンボボックス

```

from PySide6.QtWidgets import QCompleter
from PySide6.QtCore import Qt

# 大量のデータがある場合のフィルタリング
data_combo = QComboBox()
data_combo.setEditable(True)

# 大量のアイテム
large_dataset = [f"アイテム{i:03d}" for i in range(1, 1001)]
data_combo.addItems(large_dataset)

# オートコンプリート機能
completer = QCompleter(large_dataset)
completer.setCaseSensitivity(Qt.CaseInsensitive)
data_combo.setCompleter(completer)

```

スタイリングとカスタマイズ

CSS スタイルシートの例

```

combo.setStyleSheet("""
    QComboBox {
        border: 2px solid #cccccc;
        border-radius: 5px;
        padding: 5px;
        background-color: white;
        selection-background-color: #007acc;
    }
    QComboBox:hover {
        border-color: #007acc;
    }
    QComboBox::drop-down {
        subcontrol-origin: padding;
        subcontrol-position: top right;
        width: 20px;
        border-left: 1px solid #cccccc;
    }
    QComboBox::down-arrow {
        width: 10px;
        height: 10px;
    }
    QComboBox QAbstractItemView {
        border: 1px solid #cccccc;
        background-color: white;
        selection-background-color: #007acc;
    }
""")

```

ベストプラクティス

1. 適切なデフォルト選択

```
# 良い例：最も一般的な選択肢をデフォルトに
priority_combo = QComboBox()
priority_combo.addItems(["低", "中", "高", "緊急"])
priority_combo.setCurrentText("中") # 一般的な優先度をデフォルト
```

2. 空の状態の処理

```
# プレースホルダー的な項目を追加
combo.addItem("-- 選択してください --")
combo.setCurrentIndex(0)

# 実際の処理では最初の項目を除外
def get_valid_selection():
    if combo.currentIndex() > 0:
        return combo.currentText()
    return None
```

3. 大量データの効率的な処理

```
# 遅延読み込みのような機能
def populate_combo_on_demand():
    if combo.count() == 0: # まだデータが読み込まれていない
        # データベースやAPIからデータを取得
        data = fetch_data_from_source()
        combo.addItems(data)

# フォーカス時にデータを読み込み
combo.focusInEvent = lambda event: populate_combo_on_demand()
```

注意事項

- パフォーマンス: 大量のアイテムがある場合は、仮想化やページングを検討
- ユーザビリティ: 選択肢が多い場合は、検索機能やグループ化を提供
- アクセシビリティ: キーボードナビゲーションを考慮
- データ整合性: 動的にアイテムを変更する際は、現在の選択状態を適切に管理

関連するクラス

- QListWidget:** リスト表示ウィジェット
- QCompleter:** オートコンプリート機能
- QAbstractItemModel:** アイテムモデルの基底クラス
- QStandardItemModel:** 標準的なアイテムモデル

参考リンク

- [Qt公式ドキュメント - QComboBox](#)
- [PySide6公式ドキュメント](#)

QMessageBox クラス

概要

QMessageBox は、ユーザーに情報を表示したり、重要な質問への回答を求めたりするためのモーダルダイアログボックスです。情報表示、警告、エラー報告、確認ダイアログなど、様々なタイプのメッセージ表示に使用され、デスクトップアプリケーションには欠かせないコンポーネントです。

基本的な使用方法

```
from PySide6.QtWidgets import QApplication, QMessageBox, QPushButton, QWidget
import sys

app = QApplication(sys.argv)

# 基本的なメッセージボックスの表示
QMessageBox.information(None, "タイトル", "これは情報メッセージです。")

app.exec()
```

メッセージボックスの種類

QMessageBox は以下のタイプのメッセージを提供します：

タイプ	アイコン	用途	静的メソッド
Information	情報アイコン	一般的な情報の表示	QMessageBox.information()
Warning	警告アイコン	注意事項や警告	QMessageBox.warning()
Critical	エラーアイコン	エラーや重大な問題	QMessageBox.critical()
Question	質問アイコン	ユーザーへの質問	QMessageBox.question()

静的メソッドによる簡単な使用

情報表示

```
# 基本的な情報表示
QMessageBox.information(parent, "完了", "処理が正常に完了しました。")

# 戻り値を使った例
reply = QMessageBox.information(
    parent,
    "保存完了",
    "ファイルが保存されました。",
    QMessageBox.Ok
)
```

警告表示

```
# 警告メッセージ
QMessageBox.warning(parent, "警告", "この操作は元に戻せません。")

# 複数ボタンでの警告
reply = QMessageBox.warning(
```

```

parent,
"警告",
"保存されていない変更があります。続行しますか？",
QMessageBox.Yes | QMessageBox.No,
QMessageBox.No # デフォルトボタン
)

if reply == QMessageBox.Yes:
    print("続行します")
else:
    print("キャンセルしました")

```

エラー表示

```

# エラーメッセージ
QMessageBox.critical(parent, "エラー", "ファイルの読み込みに失敗しました。")

# 詳細なエラー情報
error_details = "ファイルパス: /path/to/file\nエラーコード: 404"
QMessageBox.critical(
    parent,
    "ファイルエラー",
    f"ファイルが見つかりません。{error_details}"
)

```

質問ダイアログ

```

# はい/いいえの質問
reply = QMessageBox.question(
    parent,
    "確認",
    "本当に削除しますか？",
    QMessageBox.Yes | QMessageBox.No,
    QMessageBox.No
)

if reply == QMessageBox.Yes:
    # 削除処理
    perform_delete()

```

カスタムメッセージボックス

より細かい制御が必要な場合は、QMessageBoxオブジェクトを直接作成します：

基本的なカスタムボックス

```

# カスタムメッセージボックスの作成
msg_box = QMessageBox()
msg_box.setWindowTitle("カスタム確認")
msg_box.setText("メインメッセージがここに表示されます。")
msg_box.setInformativeText("追加の詳細情報をここに表示できます。")
msg_box.setIcon(QMessageBox.Question)

# カスタムボタンの追加
msg_box.setStandardButtons(QMessageBox.Save | QMessageBox.Discard | QMessageBox.Cancel)
msg_box.setDefaultButton(QMessageBox.Save)

# 実行と結果の取得
result = msg_box.exec()

if result == QMessageBox.Save:
    print("保存が選択されました")
elif result == QMessageBox.Discard:
    print("破棄が選択されました")

```

```

else:
    print("キャンセルされました")

```

詳細な情報付きメッセージボックス

```

msg_box = QMessageBox()
msg_box.setWindowTitle("処理結果")
msg_box.setText("データの処理が完了しました。")
msg_box.setInformativeText("一部のファイルで警告が発生しました。詳細を確認しますか？")

# 詳細なテキストを設定
detailed_text = """
処理されたファイル: 150個
成功: 147個
警告: 3個
エラー: 0個

警告があったファイル:
- file1.txt: 文字エンコーディングの問題
- file2.csv: 不正な日付形式
- file3.json: 不明なフィールド
"""

msg_box.setDetailedText(detailed_text)

msg_box.setStandardButtons(QMessageBox.Ok)
msg_box.exec()

```

標準ボタンの種類

ボタン	説明	使用場面
Ok	OK	確認・了承
Cancel	キャンセル	操作の中止
Yes	はい	肯定的な回答
No	いいえ	否定的な回答
Save	保存	データの保存
Discard	破棄	変更の破棄
Apply	適用	設定の適用
Reset	リセット	初期状態に戻す
Close	閉じる	ダイアログを閉じる
Help	ヘルプ	ヘルプの表示

実用的な使用例

1. ファイル保存の確認

```

def confirm_save_file():
    if has_unsaved_changes():
        reply = QMessageBox.question(
            self,
            "未保存の変更",
            "ファイルに未保存の変更があります。保存してから続行しますか？",
            QMessageBox.Save | QMessageBox.Discard | QMessageBox.Cancel,
            QMessageBox.Save
        )

        if reply == QMessageBox.Save:
            return save_file()
        elif reply == QMessageBox.Discard:
            return True
        else: # Cancel

```

```
    return False
return True
```

2. アプリケーション終了の確認

```
def closeEvent(self, event):
    """アプリケーション終了時の確認"""
    reply = QMessageBox.question(
        self,
        "終了確認",
        "アプリケーションを終了しますか？",
        QMessageBox.Yes | QMessageBox.No,
        QMessageBox.No
    )

    if reply == QMessageBox.Yes:
        event.accept()
    else:
        event.ignore()
```

3. 操作結果の報告

```
def show_operation_result(success_count, error_count):
    if error_count == 0:
        QMessageBox.information(
            self,
            "処理完了",
            f"すべての処理が正常に完了しました。\\n処理件数: {success_count}件"
        )
    else:
        msg_box = QMessageBox()
        msg_box.setIcon(QMessageBox.Warning)
        msg_box.setWindowTitle("処理完了（警告あり）")
        msg_box.setText("処理が完了しましたが、いくつかのエラーが発生しました。")
        msg_box.setInformativeText(f"成功: {success_count}件\\nエラー: {error_count}件")
        msg_box.setStandardButtons(QMessageBox.Ok)
        msg_box.exec()
```

4. 条件付きメッセージ表示

```
def show_conditional_message(user_level, operation):
    if user_level == "beginner":
        QMessageBox.information(
            self,
            "ヒント",
            f"{operation}を実行しました。\\n\\n"
            "💡 ヒント: この機能は設定画面でカスタマイズできます。"
        )
    elif operation == "delete" and user_level != "expert":
        QMessageBox.warning(
            self,
            "削除実行",
            "アイテムが削除されました。\\n\\n"
            "⚠️ 注意: 削除されたアイテムは復元できません。"
        )
```

カスタムアイコンとスタイル

カスタムアイコンの設定

```
from PySide6.QtGui import QIcon, QPixmap

msg_box = QMessageBox()
msg_box.setWindowTitle("カスタムメッセージ")
```

```

msg_box.setText("カスタムアイコン付きのメッセージです。")

# カスタムアイコンを設定
custom_icon = QIcon("custom_icon.png")
msg_box.setIconPixmap(custom_icon pixmap(64, 64))

msg_box.exec()

```

スタイルシートの適用

```

msg_box = QMessageBox()
msg_box.setStyleSheet("""
    QMessageBox {
        background-color: #f0f0f0;
        color: #333333;
    }
    QMessageBox QPushButton {
        background-color: #007acc;
        color: white;
        border: none;
        padding: 8px 16px;
        border-radius: 4px;
        min-width: 80px;
    }
    QMessageBox QPushButton:hover {
        background-color: #005a9e;
    }
""")

```

非モーダルメッセージボックス

通常のメッセージボックスはモーダル（他の操作をブロック）ですが、非モーダルな表示も可能です：

```

def show_non_modal_message():
    msg_box = QMessageBox()
    msg_box.setWindowTitle("非モーダルメッセージ")
    msg_box.setText("この警告は他の操作をブロックしません。")
    msg_box.setIcon(QMessageBox.Information)
    msg_box.setStandardButtons(QMessageBox.Ok)

    # 非モーダルで表示
    msg_box.setModal(False)
    msg_box.show()  # exec()ではなくshow()を使用

```

ベストプラクティス

1. 適切なメッセージタイプの選択

```

# 良い例：適切なタイプを使用
QMessageBox.critical(self, "エラー", "ネットワーク接続に失敗しました。") # エラー用
QMessageBox.warning(self, "警告", "ディスク容量が不足しています。") # 警告用
QMessageBox.information(self, "完了", "バックアップが完了しました。") # 情報用

```

2. 明確で具体的なメッセージ

```

# 良い例：具体的なメッセージ
QMessageBox.question(
    self,
    "ファイル削除の確認",
    f"ファイル '{filename}' を完全に削除しますか？\n\nこの操作は元に戻せません。",
    QMessageBox.Yes | QMessageBox.No,
)

```

```
    QMessageBox.No  
)  
  
# 避けるべき例：曖昧なメッセージ  
QMessageBox.question(self, "確認", "実行しますか？")
```

3. 適切なデフォルトボタンの設定

```
# 破壊的な操作では安全なオプションをデフォルトに  
reply = QMessageBox.warning(  
    self,  
    "データ削除",  
    "すべてのデータが削除されます。続行しますか?",  
    QMessageBox.Yes | QMessageBox.No,  
    QMessageBox.No # 安全なオプションをデフォルトに  
)
```

注意事項

1. ユーザビリティ: メッセージは簡潔で分かりやすくする
2. アクセシビリティ: 適切なタイトルとアイコンを使用する
3. 國際化: テキストは翻訳可能な形で管理する
4. パフォーマンス: 頻繁に表示されるメッセージは適度に制限する

関連するクラス

- **QDialog**: カスタムダイアログの基底クラス
- **QInputDialog**: 入力を求めるダイアログ
- **QFileDialog**: ファイル選択ダイアログ
- **QColorDialog**: 色選択ダイアログ
- **QProgressDialog**: 進行状況表示ダイアログ

参考リンク

- [Qt公式ドキュメント - QMessageBox](#)
- [PySide6公式ドキュメント](#)

QGroupBox クラス

概要

`QGroupBox` は、関連するウィジェットをグループ化し、視覚的に整理するためのコンテナウィジェットです。タイトル付きの枠線でウィジェットを囲み、ユーザーインターフェースの構造を明確にします。フォーム、設定画面、オプション群の整理に広く使用されます。

基本的な使用方法

```
from PySide6.QtWidgets import QApplication, QGroupBox, QVBoxLayout, QCheckBox, QWidget
import sys

app = QApplication(sys.argv)
window = QWidget()
main_layout = QVBoxLayout()

# 基本的なQGroupBoxの作成
group_box = QGroupBox("設定オプション")
group_layout = QVBoxLayout()

# グループ内にウィジェットを追加
group_layout.addWidget(QCheckBox("オプション1"))
group_layout.addWidget(QCheckBox("オプション2"))
group_layout.addWidget(QCheckBox("オプション3"))

group_box.setLayout(group_layout)
main_layout.addWidget(group_box)

window.setLayout(main_layout)
window.show()
sys.exit(app.exec())
```

主要なプロパティとメソッド

タイトルの設定

メソッド	説明	例
<code>setTitle(title)</code>	グループボックスのタイトルを設定	<code>group_box.setTitle("新しいタイトル")</code>
<code>title()</code>	現在のタイトルを取得	<code>title = group_box.title()</code>

チェック可能なグループボックス

メソッド	説明	例
<code>setCheckable(checkable)</code>	チェック可能にする	<code>group_box.setCheckable(True)</code>
<code>isCheckable()</code>	チェック可能かどうかを確認	<code>checkable = group_box.isCheckable()</code>
<code>setChecked(checked)</code>	チェック状態を設定	<code>group_box.setChecked(True)</code>
<code>isChecked()</code>	チェック状態を取得	<code>checked = group_box.isChecked()</code>

アライメント

メソッド	説明	例
<code>setAlignment(alignment)</code>	タイトルの配置を設定	<code>group_box.setAlignment(Qt.AlignCenter)</code>
<code>alignment()</code>	現在の配置を取得	<code>align = group_box.alignment()</code>

フラット表示

メソッド	説明	例
<code>setFlat(flat)</code>	フラット表示モードを設定	<code>group_box.setFlat(True)</code>
<code>isFlat()</code>	フラット表示かどうかを確認	<code>flat = group_box.isFlat()</code>

主要なシグナル

シグナル	説明	使用例
<code>toggled(checked)</code>	チェック状態が変更された時	<code>group_box.toggled.connect(on_group_toggled)</code>
<code>clicked(checked)</code>	クリックされた時	<code>group_box.clicked.connect(on_group_clicked)</code>

実用的な使用例

1. 基本的な設定グループ

```
# 通知設定のグループ
notification_group = QGroupBox("通知設定")
notification_layout = QVBoxLayout()

email_check = QCheckBox("メール通知")
sms_check = QCheckBox("SMS通知")
push_check = QCheckBox("プッシュ通知")

notification_layout.addWidget(email_check)
notification_layout.addWidget(sms_check)
notification_layout.addWidget(push_check)

notification_group.setLayout(notification_layout)
```

2. チェック可能なグループボックス

```
# 高度な設定（有効/無効切り替え可能）
advanced_group = QGroupBox("高度な設定")
advanced_group.setCheckable(True)
advanced_group.setChecked(False) # 初期状態では無効

advanced_layout = QVBoxLayout()
debug_check = QCheckBox("デバッグモード")
verbose_check = QCheckBox("詳細ログ")
experimental_check = QCheckBox("実験的機能")

advanced_layout.addWidget(debug_check)
advanced_layout.addWidget(verbose_check)
advanced_layout.addWidget(experimental_check)

advanced_group.setLayout(advanced_layout)

# グループの有効/無効に応じて内部ウィジェットも制御
def on_advanced_toggled(checked):
    # グループが無効の場合、内部のウィジェットも無効になる
    print(f"高度な設定: {'有効' if checked else '無効'}")

advanced_group.toggled.connect(on_advanced_toggled)
```

3. 複数のグループを使ったフォーム

```
from PySide6.QtWidgets import QLineEdit, QSpinBox, QComboBox, QHBoxLayout

# ユーザー情報フォーム
main_layout = QVBoxLayout()
```

```

# 基本情報グループ
basic_group = QGroupBox("基本情報")
basic_layout = QVBoxLayout()

name_layout = QHBoxLayout()
name_layout.addWidget(QLabel("名前:"))
name_edit = QLineEdit()
name_layout.addWidget(name_edit)
basic_layout.addLayout(name_layout)

age_layout = QHBoxLayout()
age_layout.addWidget(QLabel("年齢:"))
age_spin = QSpinBox()
age_spin.setRange(0, 120)
age_layout.addWidget(age_spin)
basic_layout.addLayout(age_layout)

basic_group.setLayout(basic_layout)

# 連絡先情報グループ
contact_group = QGroupBox("連絡先情報")
contact_layout = QVBoxLayout()

email_layout = QHBoxLayout()
email_layout.addWidget(QLabel("メール:"))
email_edit = QLineEdit()
email_layout.addWidget(email_edit)
contact_layout.addLayout(email_layout)

phone_layout = QHBoxLayout()
phone_layout.addWidget(QLabel("電話:"))
phone_edit = QLineEdit()
phone_layout.addWidget(phone_edit)
contact_layout.addLayout(phone_layout)

contact_group.setLayout(contact_layout)

# 設定グループ
settings_group = QGroupBox("設定")
settings_group.setCheckable(True)
settings_layout = QVBoxLayout()

theme_layout = QHBoxLayout()
theme_layout.addWidget(QLabel("テーマ:"))
theme_combo = QComboBox()
theme_combo.addItem("ライト")
theme_combo.addItem("ダーク")
theme_combo.addItem("自動")
theme_layout.addWidget(theme_combo)
settings_layout.addLayout(theme_layout)

settings_group.setLayout(settings_layout)

# すべてのグループをメインレイアウトに追加
main_layout.addWidget(basic_group)
main_layout.addWidget(contact_group)
main_layout.addWidget(settings_group)

```

4. ラジオボタンのグループ化

```

from PySide6.QtWidgets import QRadioButton

# 優先度選択グループ
priority_group = QGroupBox("優先度")
priority_layout = QVBoxLayout()

low_radio = QRadioButton("低")
medium_radio = QRadioButton("中")
high_radio = QRadioButton("高")
urgent_radio = QRadioButton("緊急")

```

```

# デフォルト選択
medium_radio.setChecked(True)

priority_layout.addWidget(low_radio)
priority_layout.addWidget(medium_radio)
priority_layout.addWidget(high_radio)
priority_layout.addWidget(urgent_radio)

priority_group.setLayout(priority_layout)

```

5. 水平レイアウトでのグループ配置

```

# 複数のグループを水平に配置
horizontal_layout = QHBoxLayout()

# 左側のグループ
left_group = QGroupBox("入力設定")
left_layout = QVBoxLayout()
left_layout.addWidget(QCheckBox("自動保存"))
left_layout.addWidget(QCheckBox("自動補完"))
left_group.setLayout(left_layout)

# 右側のグループ
right_group = QGroupBox("表示設定")
right_layout = QVBoxLayout()
right_layout.addWidget(QCheckBox("行番号表示"))
right_layout.addWidget(QCheckBox("構文ハイライト"))
right_group.setLayout(right_layout)

horizontal_layout.addWidget(left_group)
horizontal_layout.addWidget(right_group)

```

スタイリングとカスタマイズ

CSS スタイルシートの例

```

group_box.setStyleSheet("""
    QGroupBox {
        font-weight: bold;
        border: 2px solid #cccccc;
        border-radius: 8px;
        margin-top: 10px;
        padding-top: 10px;
    }
    QGroupBox::title {
        subcontrol-origin: margin;
        left: 10px;
        padding: 0 5px 0 5px;
        color: #333333;
        background-color: white;
    }
    QGroupBox::indicator {
        width: 13px;
        height: 13px;
    }
    QGroupBox::indicator:unchecked {
        border: 1px solid #cccccc;
        background-color: white;
    }
    QGroupBox::indicator:checked {
        border: 1px solid #007acc;
        background-color: #007acc;
    }
""")

```

フラット表示のスタイル

```
# フラット表示 (枠線なし)
flat_group = QGroupBox("フラットグループ")
flat_group.setFlat(True)
flat_group.setStyleSheet("""
    QGroupBox {
        font-weight: bold;
        color: #666666;
        border: none;
        margin-top: 5px;
    }
    QGroupBox::title {
        subcontrol-origin: margin;
        left: 0px;
        padding: 0 0 5px 0;
    }
""")
""")
```

ベストプラクティス

1. 論理的なグループ化

```
# 良い例: 関連する機能をグループ化
security_group = QGroupBox("セキュリティ設定")
security_layout = QVBoxLayout()

password_check = QCheckBox("/パスワード保護")
encryption_check = QCheckBox("データ暗号化")
two_factor_check = QCheckBox("二要素認証")

security_layout.addWidget(password_check)
security_layout.addWidget(encryption_check)
security_layout.addWidget(two_factor_check)
security_group.setLayout(security_layout)
```

2. 適切なタイトルの設定

```
# 良い例: 明確で簡潔なタイトル
network_group = QGroupBox("ネットワーク設定")
appearance_group = QGroupBox("外観とテーマ")

# 避けるべき例: 暗昧なタイトル
misc_group = QGroupBox("その他") # 何の設定か不明
```

3. チェック可能グループの適切な使用

```
# 機能全体の有効/無効を制御する場合に使用
backup_group = QGroupBox("自動バックアップ")
backup_group.setCheckable(True)
backup_group.setChecked(True)

backup_layout = QVBoxLayout()
backup_layout.addWidget(QCheckBox("毎日バックアップ"))
backup_layout.addWidget(QCheckBox("クラウド同期"))

backup_group.setLayout(backup_layout)

# グループが無効になると、内部のウィジェットも自動的に無効になる
```

4. レスポンシブなレイアウト

```
# ウィンドウサイズに応じてグループの配置を調整
def create_responsive_groups():
    main_layout = QVBoxLayout()

    # 小さな画面では縦に配置
    if window_width < 800:
        main_layout.addWidget(group1)
        main_layout.addWidget(group2)
    else:
        # 大きな画面では横に配置
        horizontal_layout = QHBoxLayout()
        horizontal_layout.addWidget(group1)
        horizontal_layout.addWidget(group2)
        main_layout.addLayout(horizontal_layout)

    return main_layout
```

注意事項

1. ユーザビリティ: グループのタイトルは機能を明確に表現する
2. 視覚的階層: 重要度に応じてグループの配置を決める
3. アクセシビリティ: チェック可能グループの状態を明確に示す
4. パフォーマンス: 深いネストは避け、シンプルな構造を保つ

関連するクラス

- **QFrame**: フレーム表示の基底クラス
- **QWidget**: すべてのウィジェットの基底クラス
- **QVBoxLayout/QHBoxLayout**: レイアウト管理
- **QButtonGroup**: ボタンのグループ管理
- **QTabWidget**: タブ形式のグループ化

参考リンク

- [Qt公式ドキュメント - QGroupBox](#)
- [PySide6公式ドキュメント](#)

QVBoxLayout

QVBoxLayoutは、ウィジェットを垂直方向（上から下）に配置するレイアウトマネージャーです。

インポート

```
from PySide6.QtWidgets import QVBoxLayout
```

基本的な使用方法

```
from PySide6.QtWidgets import QVBoxLayout, QWidget

widget = QWidget()
layout = QVBoxLayout()
widget.setLayout(layout)
```

主要なメソッド

ウィジェットの追加

addWidget(widget)

```
layout.addWidget(some_widget)
```

レイアウトにウィジェットを追加します。

パラメータ:- `widget` (QWidget): 追加するウィジェット

addWidget(widget, stretch, alignment)

```
from PySide6.QtCore import Qt
layout.addWidget(widget, 1, Qt.AlignmentFlag.AlignCenter)
```

ストレッチファクターと配置を指定してウィジェットを追加します。

パラメータ:- `widget` (QWidget): 追加するウィジェット - `stretch` (int): ストレッチファクター (0以上) - `alignment` (Qt.AlignmentFlag): 配置方法

insertWidget(index, widget)

```
layout.insertWidget(0, widget)
```

指定した位置にウィジェットを挿入します。

パラメータ:- `index` (int): 挿入位置 - `widget` (QWidget): 挿入するウィジェット

レイアウトの追加

addLayout(layout)

```
sub_layout = QBoxLayout()
layout.addLayout(sub_layout)
```

レイアウト内に別のレイアウトを追加します。

パラメータ:- `layout` (QLayout): 追加するレイアウト

スペースの管理

addStretch(stretch=0)

```
layout.addStretch()      # デフォルトのストレッチ
layout.addStretch(2)    # ストレッチファクター2
```

レイアウトに伸縮可能なスペースを追加します。

パラメータ:- `stretch` (int): ストレッチファクター

addSpacing(size)

```
layout.addSpacing(20)  # 20ピクセルの固定スペース
```

固定サイズのスペースを追加します。

パラメータ:- `size` (int): スペースのサイズ (ピクセル)

ウィジェットの削除

removeWidget(widget)

```
layout.removeWidget(some_widget)
```

レイアウトからウィジェットを削除します。

パラメータ:- `widget` (QWidget): 削除するウィジェット

マージンとスペーシング

setContentsMargins(left, top, right, bottom)

```
layout.setContentsMargins(10, 10, 10, 10)
```

レイアウトの外側マージンを設定します。

パラメータ:- `left` (int): 左マージン - `top` (int): 上マージン - `right` (int): 右マージン - `bottom` (int): 下マージン

setSpacing(spacing)

```
layout.setSpacing(5)
```

ウィジェット間のスペーシングを設定します。

パラメータ:- `spacing` (int): スペーシングのサイズ (ピクセル)

情報取得

count()

```
widget_count = layout.count()
```

レイアウト内のアイテム数を取得します。

戻り値: int - アイテム数

itemAt(index)

```
item = layout.itemAt(0)
```

指定したインデックスのレイアウトアイテムを取得します。

パラメータ:- `index` (int): インデックス

戻り値: QLayoutItem - レイアウトアイテム

使用例

基本的な垂直レイアウト

```
import sys
from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel, QPushButton

class MainWindow(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("垂直レイアウト例")

        # レイアウトの作成
        layout = QVBoxLayout()
        self.setLayout(layout)

        # ウィジェットを順番に追加
        layout.addWidget(QLabel("ラベル1"))
        layout.addWidget(QLabel("ラベル2"))
        layout.addWidget(QPushButton("ボタン1"))
        layout.addWidget(QPushButton("ボタン2"))

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec())
```

ストレッチとスペーシングの使用

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QLabel, QPushButton

class StretchLayout(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        # 上部のウィジェット
        layout.addWidget(QLabel("上部"))

        # 伸縮可能なスペース
        layout.addStretch(1)

        # 中央のウィジェット
        layout.addWidget(QPushButton("中央ボタン"))

        # より大きなストレッチ
        layout.addStretch(2)

        # 下部のウィジェット
        layout.addWidget(QLabel("下部"))

        # マージンとスペーシングの設定
        layout.setContentsMargins(20, 20, 20, 20)
        layout.setSpacing(10)
```

動的なウィジェット追加/削除

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QPushButton, QLabel

class DynamicLayout(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("動的レイアウト")
        self.counter = 0

        self.layout = QVBoxLayout()
        self.setLayout(self.layout)

        # コントロールボタン
        add_button = QPushButton("ラベル追加")
        add_button.clicked.connect(self.add_label)
        self.layout.addWidget(add_button)

        remove_button = QPushButton("最後のラベル削除")
        remove_button.clicked.connect(self.remove_label)
        self.layout.addWidget(remove_button)

        # 区切り線
        self.layout.addSpacing(20)

        # 動的ラベル用のリスト
        self.labels = []

    def add_label(self):
        self.counter += 1
        label = QLabel(f"動的ラベル {self.counter}")
        self.labels.append(label)
        self.layout.addWidget(label)

    def remove_label(self):
        if self.labels:
            label = self.labels.pop()
```

```
    self.layout.removeWidget(label)
    label.deleteLater() # メモリから削除
```

ネストしたレイアウト

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QPushButton, QLabel

class NestedLayout(QWidget):
    def __init__(self):
        super().__init__()
        # メインの垂直レイアウト
        main_layout = QVBoxLayout()
        self.setLayout(main_layout)

        # タイトル
        title = QLabel("ネストしたレイアウトの例")
        main_layout.addWidget(title)

        # 水平レイアウトを作成
        horizontal_layout = QHBoxLayout()
        horizontal_layout.addWidget(QPushButton("左ボタン"))
        horizontal_layout.addWidget(QPushButton("右ボタン"))

        # 水平レイアウトを垂直レイアウトに追加
        main_layout.addLayout(horizontal_layout)

        # 下部のボタン
        main_layout.addWidget(QPushButton("下部ボタン"))
```

配置とストレッチの理解

ストレッチファクター

```
# ストレッチファクターの例
layout.addWidget(widget1, 1) # 1の比率
layout.addWidget(widget2, 2) # 2の比率 (widget1の2倍のスペースを取る)
layout.addWidget(widget3, 1) # 1の比率
```

配置オプション

```
from PySide6.QtCore import Qt

# 中央揃え
layout.addWidget(widget, 0, Qt.AlignmentFlag.AlignCenter)

# 左揃え
layout.addWidget(widget, 0, Qt.AlignmentFlag.AlignLeft)

# 右揃え
layout.addWidget(widget, 0, Qt.AlignmentFlag.AlignRight)
```

注意事項

- レイアウトは必ずウィジェットに設定してから使用してください
- `removeWidget()` した後は `deleteLater()` を呼んでメモリを解放することを推奨します
- ストレッチファクターが0の場合、ウィジェットは最小サイズを保持します
- マージンとスペーシングは見た目に大きく影響するため、適切に設定してください

QHBoxLayout クラス

概要

`QHBoxLayout` は、ウィジェットを水平方向（横方向）に配置するレイアウトマネージャーです。`QVBoxLayout`と対をなす基本的なレイアウトクラスで、ウィジェットを左から右に順番に配置します。ツールバー、ボタン配列、水平なコントロール群の作成に最適です。

基本的な使用方法

```
from PySide6.QtWidgets import QApplication, QHBoxLayout, QPushButton, QWidget
import sys

app = QApplication(sys.argv)
window = QWidget()

# 水平レイアウトの作成
layout = QHBoxLayout()

# ボタンを追加
layout.addWidget(QPushButton("ボタン1"))
layout.addWidget(QPushButton("ボタン2"))
layout.addWidget(QPushButton("ボタン3"))

window.setLayout(layout)
window.show()
sys.exit(app.exec())
```

主要なメソッド

ウィジェットの追加・削除

メソッド	説明	例
<code>addWidget(widget, stretch)</code>	ウィジェットを追加	<code>layout.addWidget(button, 1)</code>
<code>addLayout(layout, stretch)</code>	レイアウトを追加	<code>layout.addLayout(sub_layout)</code>
<code>insertWidget(index, widget)</code>	指定位置にウィジェットを挿入	<code>layout.insertWidget(1, button)</code>
<code>removeWidget(widget)</code>	ウィジェットを削除	<code>layout.removeWidget(button)</code>
<code>addStretch(stretch)</code>	伸縮可能なスペースを追加	<code>layout.addStretch(1)</code>
<code>addSpacing(size)</code>	固定サイズのスペースを追加	<code>layout.addSpacing(20)</code>

スペーシングとマージン

メソッド	説明	例
<code>setSpacing(spacing)</code>	ウィジェット間のスペースを設定	<code>layout.setSpacing(10)</code>
<code>spacing()</code>	現在のスペーシングを取得	<code>space = layout.spacing()</code>
<code>setContentsMargins(left, top, right, bottom)</code>	マージンを設定	<code>layout.setContentsMargins(10, 10, 10, 10)</code>
<code>contentsMargins()</code>	現在のマージンを取得	<code>margins = layout.contentsMargins()</code>

ストレッチファクター

メソッド	説明	例
<code>setStretchFactor(widget, stretch)</code>	ウィジェットのストレッチファクターを設定	<code>layout.setStretchFactor(widget, 2)</code>
<code>setStretchFactor(layout, stretch)</code>	レイアウトのストレッチファクターを設定	<code>layout.setStretchFactor(sub_layout, 1)</code>

実用的な使用例

1. ボタン配列の作成

```
button_layout = QHBoxLayout()

# ボタンを追加
ok_button = QPushButton("OK")
cancel_button = QPushButton("キャンセル")
apply_button = QPushButton("適用")

# 左側にスペースを追加してボタンを右寄せ
button_layout.addStretch()
button_layout.addWidget(ok_button)
button_layout.addWidget(cancel_button)
button_layout.addWidget(apply_button)
```

2. ラベルと入力フィールドの組み合わせ

```
from PySide6.QtWidgets import QLabel, QLineEdit

form_layout = QHBoxLayout()

# ラベルと入力フィールド
name_label = QLabel("名前:")
name_edit = QLineEdit()

form_layout.addWidget(name_label)
form_layout.addWidget(name_edit, 1) # ストレッチファクター1で拡張
```

3. ツールバー風レイアウト

```
toolbar_layout = QHBoxLayout()

# ツールボタン
new_button = QPushButton("新規")
open_button = QPushButton("開く")
save_button = QPushButton("保存")

toolbar_layout.addWidget(new_button)
toolbar_layout.addWidget(open_button)
toolbar_layout.addWidget(save_button)
toolbar_layout.addSpacing(20) # セパレーター的なスペース

# 右側に検索ボックス
search_edit = QLineEdit()
search_edit.setPlaceholderText("検索...")
toolbar_layout.addStretch() # 左右に分離
toolbar_layout.addWidget(search_edit)
```

4. 複数レイアウトの組み合わせ

```
# メインレイアウト（垂直）
main_layout = QVBoxLayout()

# ヘッダー（水平）
header_layout = QHBoxLayout()
title_label = QLabel("アプリケーション")
close_button = QPushButton("x")
header_layout.addWidget(title_label)
header_layout.addStretch()
header_layout.addWidget(close_button)

# コンテンツエリア
content_area = QTextEdit()

# フッター（水平）
footer_layout = QHBoxLayout()
status_label = QLabel("準備完了")
progress_bar = QProgressBar()
footer_layout.addWidget(status_label)
footer_layout.addWidget(progress_bar, 1)

# すべてを結合
main_layout.addLayout(header_layout)
main_layout.addWidget(content_area, 1) # メインコンテンツが拡張
main_layout.addLayout(footer_layout)
```

QVBoxLayoutとの比較

特徴	QHBoxLayout	QVBoxLayout
配置方向	水平（左→右）	垂直（上→下）
主要用途	ボタン配列、ツールバー	フォーム、メインレイアウト
拡張方向	垂直方向に拡張	水平方向に拡張

ストレッチファクターの活用

ストレッチファクターは、余剰スペースをどのように分配するかを制御します：

```
layout = QHBoxLayout()

# 固定サイズボタン
button1 = QPushButton("固定")
button2 = QPushButton("拡張1")
button3 = QPushButton("拡張2")

layout.addWidget(button1)           # ストレッチファクター 0 (固定)
layout.addWidget(button2, 1)         # ストレッチファクター 1
layout.addWidget(button3, 2)         # ストレッチファクター 2 (button2の2倍拡張)

# button2とbutton3は、2:4の比率で余剰スペースを分割
```

レイアウトの入れ子構造

複雑なUIは、複数のレイアウトを組み合わせて構築します：

```

# 外側の垂直レイアウト
outer_layout = QVBoxLayout()

# 上部の水平レイアウト
top_layout = QHBoxLayout()
top_layout.addWidget(QLabel("左"))
top_layout.addWidget(QLabel("右"))

# 中央の水平レイアウト
center_layout = QHBoxLayout()
center_layout.addWidget(QLineEdit("左入力"))
center_layout.addWidget(QLineEdit("右入力"))

# 下部の水平レイアウト
bottom_layout = QHBoxLayout()
bottom_layout.addStretch()
bottom_layout.addWidget(QPushButton("OK"))
bottom_layout.addWidget(QPushButton("キャンセル"))

# すべてを結合
outer_layout.addLayout(top_layout)
outer_layout.addLayout(center_layout)
outer_layout.addLayout(bottom_layout)

```

スペーシングとマージンの調整

視覚的に美しいレイアウトを作るためのテクニック：

```

layout = QHBoxLayout()

# 全体のマージンを設定
layout.setContentsMargins(20, 10, 20, 10) # 左、上、右、下

# ウィジエット間のスペーシングを設定
layout.setSpacing(15)

# 個別にスペースを制御
layout.addWidget(QPushButton("ボタン1"))
layout.addSpacing(30) # 固定スペース
layout.addWidget(QPushButton("ボタン2"))
layout.addStretch() # 可変スペース
layout.addWidget(QPushButton("ボタン3"))

```

ベストプラクティス

1. 適切なストレッチファクターの使用

```

# 良い例：メイン要素が拡張し、サブ要素は固定
layout.addWidget(sidebar) # 固定幅
layout.addWidget(main_content, 1) # 拡張
layout.addWidget(properties) # 固定幅

```

2. 視覚的なグループ化

```

# 関連するコントロールをグループ化
controls_layout = QHBoxLayout()
controls_layout.addWidget(QLabel("設定："))
controls_layout.addWidget(QSpinBox())
controls_layout.addWidget(QCheckBox("有効"))

# セパレーターでグループを分離

```

```
main_layout.addLayout(controls_layout)
main_layout.addSpacing(20) # グループ間のスペース
```

3. レスポンシブデザイン

```
# ウィンドウサイズに応じて調整されるレイアウト
responsive_layout = QHBoxLayout()
responsive_layout.addWidget(fixed_sidebar, 0)      # 固定サイズ
responsive_layout.addWidget(flexible_content, 1)    # 可変サイズ
```

注意事項

1. **パフォーマンス**: 深い入れ子構造は避ける
2. **可読性**: 複雑なレイアウトは論理的に分割する
3. **保守性**: レイアウトの構築ロジックを関数化する
4. **アクセシビリティ**: タブオーダーを考慮した配置にする

関連するクラス

- **QVBoxLayout**: 垂直方向のレイアウト
- **QGridLayout**: グリッド形式のレイアウト
- **QFormLayout**: フォーム形式のレイアウト
- **QStackedLayout**: 重ねて表示するレイアウト
- **QSplitter**: ユーザーがサイズ調整可能な分割レイアウト

参考リンク

- [Qt公式ドキュメント - QVBoxLayout](#)
- [Qt公式ドキュメント - Layout Management](#)
- [PySide6公式ドキュメント](#)

QTableWidget

QTableWidgetは、テーブル形式のデータを表示・編集するためのウィジェットです。pandasのDataFrameを表示するのに非常に適しています。

インポート

```
from PySide6.QtWidgets import QTableWidget, QTableWidgetItem
```

基本的な使用方法

```
from PySide6.QtWidgets import QTableWidget, QTableWidgetItem

table = QTableWidget(3, 4) # 3行4列のテーブル
table.setHorizontalHeaderLabels(['列1', '列2', '列3', '列4'])
```

主要なメソッド

テーブル構造の設定

rowCount(rows)

```
table.setRowCount(10)
```

テーブルの行数を設定します。

パラメータ	型	説明
rows	int	行数

setColumnCount(columns)

```
table.setColumnCount(5)
```

テーブルの列数を設定します。

パラメータ	型	説明
columns	int	列数

rowCount()

```
rows = table.rowCount()
```

現在の行数を取得します。

戻り値: int - 行数

columnCount()

```
cols = table.columnCount()
```

現在の列数を取得します。

戻り値: int - 列数

ヘッダーの設定

setHorizontalHeaderLabels(labels)

```
table.setHorizontalHeaderLabels(['名前', '年齢', '職業'])
```

水平ヘッダー（列ヘッダー）のラベルを設定します。

パラメータ	型	説明
labels	list[str]	ヘッダーラベルのリスト

setVerticalHeaderLabels(labels)

```
table.setVerticalHeaderLabels(['1行目', '2行目', '3行目'])
```

垂直ヘッダー（行ヘッダー）のラベルを設定します。

パラメータ	型	説明
labels	list[str]	ヘッダーラベルのリスト

データの設定と取得

setItem(row, column, item)

```
item = QTableWidgetItem("値")
table.setItem(0, 0, item)
```

指定したセルにアイテムを設定します。

パラメータ	型	説明
row	int	行インデックス
column	int	列インデックス
item	QTableWidgetItem	設定するアイテム

item(row, column)

```
item = table.item(0, 0)
if item:
    text = item.text()
```

指定したセルのアイテムを取得します。

パラメータ	型	説明
row	int	行インデックス
column	int	列インデックス

戻り値: QTableWidgetItem - セルのアイテム（Noneの場合もあり）

セルの編集制御

setEditTriggers(triggers)

```
from PySide6.QtWidgets import QAbstractItemView
table.setEditTriggers(QAbstractItemView.EditTrigger.NoEditTriggers)
```

セルの編集トリガーを設定します。

パラメータ	型	説明
triggers	QAbstractItemView.EditTrigger	編集トリガー

選択動作の設定

setSelectionBehavior(behavior)

```
from PySide6.QtWidgets import QAbstractItemView
table.setSelectionBehavior(QAbstractItemView.SelectionBehavior.SelectRows)
```

選択動作を設定します。

パラメータ	型	説明
behavior	QAbstractItemView.SelectionBehavior	選択動作

setSelectionMode(mode)

```
from PySide6.QtWidgets import QAbstractItemView
table.setSelectionMode(QAbstractItemView.SelectionMode.SingleSelection)
```

選択モードを設定します。

パラメータ	型	説明
mode	QAbstractItemView.SelectionMode	選択モード

列幅の調整

resizeColumnsToContents()

```
table.resizeColumnsToContents()
```

すべての列を内容に合わせてリサイズします。

resizeColumnToContents(column)

```
table.resizeColumnToContents(0)
```

指定した列を内容に合わせてリサイズします。

パラメータ	型	説明
column	int	列インデックス

setColumnWidth(column, width)

```
table.setColumnWidth(0, 100)
```

指定した列の幅を設定します。

パラメータ	型	説明
column	int	列インデックス
width	int	幅 (ピクセル)

pandas DataFrameとの連携

DataFrameを表示する関数

```
import pandas as pd
```

```

from PySide6.QtWidgets import QTableWidget, QTableWidgetItem
from PySide6.QtCore import Qt

def display_dataframe(table_widget, dataframe):
    """
    pandas DataFrameをQTableWidgetに表示する
    """
    # テーブルサイズの設定
    table_widget.setRowCount(len(dataframe))
    table_widget.setColumnCount(len(dataframe.columns))

    # ヘッダーの設定
    table_widget.setHorizontalHeaderLabels(dataframe.columns.tolist())
    table_widget.setVerticalHeaderLabels([str(i) for i in dataframe.index])

    # データの設定
    for row in range(len(dataframe)):
        for col in range(len(dataframe.columns)):
            value = dataframe.iloc[row, col]
            item = QTableWidgetItem(str(value))

            # データ型に応じてアライメントを設定
            if pd.api.types.is_numeric_dtype(dataframe.dtypes[col]):
                item.setTextAlignment(Qt.AlignmentFlag.AlignRight | Qt.AlignmentFlag.AlignVCenter)
            else:
                item.setTextAlignment(Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignVCenter)

            # 編集不可に設定
            item.setFlags(item.flags() & ~Qt.ItemFlag.ItemIsEditable)

            table_widget.setItem(row, col, item)

    # 列幅を内容に合わせて調整
    table_widget.resizeColumnsToContents()

def get_dataframe_from_table(table_widget):
    """
    QTableWidgetからpandas DataFrameを取得する
    """
    # ヘッダーラベルの取得
    columns = []
    for col in range(table_widget.columnCount()):
        header_item = table_widget.horizontalHeaderItem(col)
        columns.append(header_item.text() if header_item else f"Column_{col}")

    # データの取得
    data = []
    for row in range(table_widget.rowCount()):
        row_data = []
        for col in range(table_widget.columnCount()):
            item = table_widget.item(row, col)
            row_data.append(item.text() if item else "")
        data.append(row_data)

    return pd.DataFrame(data, columns=columns)

```

使用例

基本的なテーブル

```

import sys
import pandas as pd
from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QTableWidget, QTableWidgetItem

class DataFrameViewer(QWidget):
    def __init__(self):
        super().__init__()
        self.setWindowTitle("DataFrame Viewer")
        self.setGeometry(100, 100, 600, 400)

        layout = QVBoxLayout()
        self.setLayout(layout)

        # サンプルデータの作成
        self.df = pd.DataFrame({
            '名前': ['田中', '佐藤', '鈴木'],
            '年齢': [25, 30, 28],
        })

```

```

        '職業': ['エンジニア', 'デザイナー', 'マネージャー'],
        '給与': [500000, 450000, 600000]
    })

    # テーブルウィジェットの作成
    self.table = QTableWidget()
    layout.addWidget(self.table)

    # DataFrameを表示
    display_dataframe(self.table, self.df)

if __name__ == "__main__":
    app = QApplication(sys.argv)
    window = DataFrameViewer()
    window.show()
    sys.exit(app.exec())

```

フィルタリング機能付きテーブル

```

from PySide6.QtWidgets import QWidget, QVBoxLayout, QHBoxLayout, QLineEdit, QPushButton, QTableWidget
import pandas as pd

class FilterableDataFrameViewer(QWidget):
    def __init__(self, dataframe):
        super().__init__()
        self.original_df = dataframe
        self.filtered_df = dataframe.copy()

        self.setWindowTitle("フィルタリング可能なDataFrame Viewer")
        self.setGeometry(100, 100, 800, 600)

        # レイアウトの設定
        main_layout = QVBoxLayout()
        self.setLayout(main_layout)

        # フィルター用のUI
        filter_layout = QHBoxLayout()
        self.filter_input = QLineEdit()
        self.filter_input.setPlaceholderText("フィルター条件を入力 (例: 年齢 > 25) ")
        filter_button = QPushButton("フィルター実行")
        reset_button = QPushButton("リセット")

        filter_layout.addWidget(self.filter_input)
        filter_layout.addWidget(filter_button)
        filter_layout.addWidget(reset_button)

        # テーブル
        self.table = QTableWidget()

        main_layout.addLayout(filter_layout)
        main_layout.addWidget(self.table)

        # イベント接続
        filter_button.clicked.connect(self.apply_filter)
        reset_button.clicked.connect(self.reset_filter)

        # 初期表示
        display_dataframe(self.table, self.filtered_df)

    def apply_filter(self):
        filter_text = self.filter_input.text().strip()
        if not filter_text:
            return

        try:
            # 簡単なフィルタリング (実際のアプリケーションではより安全な方法を使用してください)
            self.filtered_df = self.original_df.query(filter_text)
            display_dataframe(self.table, self.filtered_df)
        except Exception as e:
            print(f"フィルターエラー: {e}")

    def reset_filter(self):
        self.filtered_df = self.original_df.copy()
        self.filter_input.clear()
        display_dataframe(self.table, self.filtered_df)

```

カスタムセルスタイル

```
from PySide6.QtWidgets import QTableWidget, QTableWidgetItem
from PySide6.QtGui import QColor
from PySide6.QtCore import Qt

def display_styled_dataframe(table_widget, dataframe):
    """
    スタイル付きでDataFrameを表示
    """
    display_dataframe(table_widget, dataframe)

    # 数値列の背景色を変更
    for col in range(len(dataframe.columns)):
        if pd.api.types.is_numeric_dtype(dataframe.dtypes[col]):
            for row in range(len(dataframe)):
                item = table_widget.item(row, col)
                if item:
                    # 数値の大きさに応じて色を変更
                    value = float(dataframe.iloc[row, col])
                    max_val = dataframe.iloc[:, col].max()
                    min_val = dataframe.iloc[:, col].min()

                    if max_val != min_val:
                        ratio = (value - min_val) / (max_val - min_val)
                        # 薄い青から濃い青へのグラデーション
                        color = QColor(200 + int(55 * (1 - ratio)), 220 + int(35 * (1 - ratio)), 255)
                        item.setBackground(color)
```

選択動作の定数

SelectionBehavior

定数	説明
QAbstractItemView.SelectionBehavior.SelectItems	アイテム単位で選択
QAbstractItemView.SelectionBehavior.SelectRows	行単位で選択
QAbstractItemView.SelectionBehavior.SelectColumns	列単位で選択

SelectionMode

定数	説明
QAbstractItemView.SelectionMode.NoSelection	選択不可
QAbstractItemView.SelectionMode.SingleSelection	単一選択
QAbstractItemView.SelectionMode.MultiSelection	複数選択
QAbstractItemView.SelectionMode.ExtendedSelection	拡張選択 (Ctrl+クリックなど)

EditTrigger

定数	説明
QAbstractItemView.EditTrigger.NoEditTriggers	編集不可
QAbstractItemView.EditTrigger.CurrentChanged	選択変更時
QAbstractItemView.EditTrigger.DoubleClicked	ダブルクリック時
QAbstractItemView.EditTrigger.SelectedClicked	選択済みアイテムクリック時
QAbstractItemView.EditTrigger.EditKeyPressed	編集キー押下時
QAbstractItemView.EditTrigger.AnyKeyPressed	任意のキー押下時
QAbstractItemView.EditTrigger.AllEditTriggers	すべてのトリガー

注意事項

- 大量のデータを扱う場合は、QTableViewとモデルベースのアプローチを検討してください
- セルアイテムは必要に応じて編集可能/不可を設定してください
- pandasのDataFrameとの相互変換時は、データ型の変換に注意してください
- パフォーマンスが重要な場合は、仮想化されたビューの使用を検討してください

QFont

QFontは、フォントの種類、サイズ、スタイルなどを定義するクラスです。ウィジェットのテキスト表示フォントを設定する際に使用します。

インポート

```
from PySide6.QtGui import QFont
```

基本的な使用方法

```
from PySide6.QtGui import QFont

font = QFont("Arial", 12)
widget.setFont(font)
```

主要なメソッド

コンストラクタ

QFont()

```
font = QFont() # デフォルトフォント
```

デフォルトフォントを作成します。

QFont(family, pointSize)

```
font = QFont("Arial", 14)
```

フォントファミリーとサイズを指定してフォントを作成します。

パラメータ	型	説明
family	str	フォントファミリー名
pointSize	int	フォントサイズ（ポイント）

QFont(family, pointSize, weight)

```
font = QFont("Arial", 14, QFont.Weight.Bold)
```

フォントファミリー、サイズ、ウェイトを指定してフォントを作成します。

パラメータ	型	説明
family	str	フォントファミリー名
pointSize	int	フォントサイズ（ポイント）
weight	QFont.Weight	フォントの太さ

フォント設定

setFontFamily(family)

```
font.setFontFamily("Times New Roman")
```

フォントファミリーを設定します。

パラメータ	型	説明
family	str	フォントファミリーネ名

setPointSize(pointSize)

```
font.setPointSize(16)
```

フォントサイズを設定します。

パラメータ	型	説明
pointSize	int	フォントサイズ（ポイント）

setWeight(weight)

```
font.setWeight(QFont.Weight.Bold)
```

フォントの太さを設定します。

パラメータ	型	説明
weight	QFont.Weight	フォントの太さ

setBold(bold)

```
font.setBold(True)
```

太字の有効/無効を設定します。

パラメータ	型	説明
bold	bool	True で太字

setItalic(italic)

```
font.setItalic(True)
```

斜体の有効/無効を設定します。

パラメータ	型	説明
italic	bool	True で斜体

setUnderline(underline)

```
font.setUnderline(True)
```

下線の有効/無効を設定します。

パラメータ	型	説明
underline	bool	True で下線

フォント情報取得

family()

```
family_name = font.family()
```

フォントファミリー名を取得します。

戻り値: str - フォントファミリー名

pointSize()

```
size = font.pointSize()
```

フォントサイズを取得します。

戻り値: int - フォントサイズ (ポイント)

weight()

```
weight = font.weight()
```

フォントの太さを取得します。

戻り値: QFont.Weight - フォントの太さ

bold()

```
is_bold = font.bold()
```

太字かどうかを確認します。

戻り値: bool - 太字の場合True

italic()

```
is_italic = font.italic()
```

斜体かどうかを確認します。

戻り値: bool - 斜体の場合True

使用例

基本的なフォント設定

```

import sys
from PySide6.QtWidgets import QApplication, QWidget, QVBoxLayout, QLabel
from PySide6.QtGui import QFont

class FontExample(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        # 標準フォント
        label1 = QLabel("標準フォント")
        layout.addWidget(label1)

        # カスタムフォント
        label2 = QLabel("カスタムフォント")
        custom_font = QFont("Arial", 16, QFont.Weight.Bold)
        label2.setFont(custom_font)
        layout.addWidget(label2)

        # 斜体フォント
        label3 = QLabel("斜体フォント")
        italic_font = QFont("Times", 14)
        italic_font.setItalic(True)
        label3.setFont(italic_font)
        layout.addWidget(label3)

    if __name__ == "__main__":
        app = QApplication(sys.argv)
        window = FontExample()
        window.show()
        sys.exit(app.exec())

```

様々なフォントスタイル

```

from PySide6.QtWidgets import QWidget, QVBoxLayout, QLabel
from PySide6.QtGui import QFont

class FontStyles(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        font_examples = [
            ("細字フォント", QFont("Arial", 12, QFont.Weight.Light)),
            ("標準フォント", QFont("Arial", 12, QFont.Weight.Normal)),
            ("太字フォント", QFont("Arial", 12, QFont.Weight.Bold)),
            ("極太フォント", QFont("Arial", 12, QFont.Weight.Black)),
        ]

        for text, font in font_examples:
            label = QLabel(text)
            label.setFont(font)
            layout.addWidget(label)

        # 組み合わせ例
        combo_label = QLabel("太字+斜体+下線")
        combo_font = QFont("Arial", 14)
        combo_font.setBold(True)
        combo_font.setItalic(True)
        combo_font.setUnderline(True)
        combo_label.setFont(combo_font)
        layout.addWidget(combo_label)

```

動的フォント変更

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QLabel, QPushButton, QSlider
from PySide6.QtGui import QFont
from PySide6.QtCore import Qt

class DynamicFont(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        # テキストラベル
        self.text_label = QLabel("サンプルテキスト")
        self.current_font = QFont("Arial", 12)
        self.text_label.setFont(self.current_font)
        layout.addWidget(self.text_label)

        # フォントサイズスライダー
        self.size_slider = QSlider(Qt.Orientation.Horizontal)
        self.size_slider.setMinimum(8)
        self.size_slider.setMaximum(48)
        self.size_slider.setValue(12)
        self.size_slider.valueChanged.connect(self.change_font_size)
        layout.addWidget(self.size_slider)

        # 太字トグル
        bold_button = QPushButton("太字切り替え")
        bold_button.setCheckable(True)
        bold_button.clicked.connect(self.toggle_bold)
        layout.addWidget(bold_button)

        # 斜体トグル
        italic_button = QPushButton("斜体切り替え")
        italic_button.setCheckable(True)
        italic_button.clicked.connect(self.toggle_italic)
        layout.addWidget(italic_button)

    def change_font_size(self, size):
        self.current_font.setPointSize(size)
        self.text_label.setFont(self.current_font)

    def toggle_bold(self, checked):
        self.current_font.setBold(checked)
        self.text_label.setFont(self.current_font)

    def toggle_italic(self, checked):
        self.current_font.setItalic(checked)
        self.text_label.setFont(self.current_font)
```

日本語フォント設定

```
from PySide6.QtWidgets import QWidget, QVBoxLayout, QLabel
from PySide6.QtGui import QFont

class JapaneseFonts(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        # 日本語フォントの例
        japanese_fonts = [
            ("メイリオ", "こんにちは、世界！"),
            ("MS ゴシック", "こんにちは、世界！"),
            ("MS 明朝", "こんにちは、世界！"),
            ("游ゴシック", "こんにちは、世界！"),
        ]
```

```

for font_name, text in japanese_fonts:
    label = QLabel(f"{font_name}: {text}")
    font = QFont(font_name, 14)
    label.setFont(font)
    layout.addWidget(label)

```

フォントウェイト定数

定数	値	説明
QFont.Weight.Thin	100	極細
QFont.Weight.ExtraLight	200	極細
QFont.Weight.Light	300	細字
QFont.Weight.Normal	400	標準（デフォルト）
QFont.Weight.Medium	500	中字
QFont.Weight.DemiBold	600	準太字
QFont.Weight.Bold	700	太字
QFont.Weight.ExtraBold	800	極太字
QFont.Weight.Black	900	最太字

よく使用されるフォントファミリー

Windows

- "Arial"
- "Times New Roman"
- "Courier New"
- "Verdana"
- "メイリオ"
- "MS ゴシック"
- "MS 明朝"

macOS

- "Helvetica"
- "Times"
- "Courier"
- "Hiragino Sans"
- "Hiragino Mincho ProN"

Linux

- "Liberation Sans"
- "Liberation Serif"
- "Liberation Mono"
- "DejaVu Sans"
- "Noto Sans CJK JP"

注意事項

- フォントファミリーが存在しない場合、システムのデフォルトフォントが使用されます
- 日本語テキストを表示する場合は、日本語対応フォントを指定してください
- フォントサイズは通常8~72ポイントの範囲で設定します

- `setBold(True)` と `setWeight(QFont.Weight.Bold)` は同じ効果があります

QColor リファレンス

PySide6における色の定義と操作についてのリファレンス資料です。

概要

`QColor` クラスは、色を表現し操作するためのクラスです。RGB、HSV、CMYK、HSLなど様々な色空間をサポートします。

サンプルアプリケーション

このリファレンスには、3つの実践的なサンプルアプリケーションが含まれています：

1. 基本的な色の作成 (`qcolor_basic.py`)

- RGB値、16進数、色名からの色の作成方法
- 各種色作成方法の比較とデモンストレーション
- コンパクトなウィンドウサイズ (600x400)

2. RGB値の制御 (`qcolor_rgb_control.py`)

- スライダーとスピンドルによるRGB値の調整
- リアルタイムな色プレビュー
- RGB値と16進数の表示

3. カラーパレットとテーマ (`qcolor_palette.py`)

- Material Designカラーパレットの表示
- ライト/ダークテーマの切り替え機能
- 美しい色の組み合わせの実例

基本的な使用方法

色の作成

```
from PySide6.QtGui import QColor

# RGB値から作成
color1 = QColor(255, 0, 0) # 赤
color2 = QColor(0, 255, 0) # 緑
color3 = QColor(0, 0, 255) # 青

# 16進数から作成
color4 = QColor("#FF0000") # 赤
color5 = QColor("#00FF00") # 緑

# 色名から作成
color6 = QColor("red")
color7 = QColor("blue")
```

カラーサンプルウィジェットの作成

以下は実際のサンプルアプリケーションで使用されている色サンプルの作成方法です：

```

def create_color_sample(self, color, text):
    """色サンプルウィジェットの作成"""
    widget = QWidget()
    widget.setFixedSize(80, 60)

    # 色に基づいてテキストの色を決定
    luminance = (0.299 * color.red() + 0.587 * color.green() + 0.114 * color.blue()) / 255
    text_color = "white" if luminance < 0.5 else "black"

    widget.setStyleSheet(f"""
        QWidget {{
            background-color: {color.name()};
            border: 1px solid #333;
            border-radius: 4px;
        }}
    """) 

    layout = QVBoxLayout()
    label = QLabel(text)
    label.setAlignment(Qt.AlignmentFlag.AlignCenter)
    label.setStyleSheet(f"color: {text_color}; font-size: 10px; font-weight: bold;")
    label.setWordWrap(True)
    layout.addWidget(label)
    widget.setLayout(layout)

    return widget

```

RGB値の制御

スライダーによる色の制御

```

# RGB スライダーの設定例
self.r_slider = QSlider(Qt.Orientation.Vertical)
self.r_slider.setRange(0, 255)
self.r_slider.setValue(128)
self.r_slider.valueChanged.connect(self.update_rgb_color)

def update_rgb_color(self):
    """RGB値の変更に基づいて色を更新"""
    r = self.r_slider.value()
    g = self.g_slider.value()
    b = self.b_slider.value()

    self.update_color_display(r, g, b)

def update_color_display(self, r, g, b):
    """色表示の更新"""
    color = QColor(r, g, b)

    # 背景色を更新
    self.color_display.setStyleSheet(f"""
        QLabel {{
            border: 2px solid black;
            background-color: rgb({r}, {g}, {b});
            color: {'white' if (0.299 * r + 0.587 * g + 0.114 * b) < 128 else 'black'};
            font-weight: bold;
        }}
    """) 

    # 色情報を更新
    self.color_info.setText(f"RGB({r}, {g}, {b})\n{color.name().upper()}")

```

色空間の変換

RGB値の取得・設定

```
color = QColor(255, 128, 64)

# RGB値の取得
r = color.red()
g = color.green()
b = color.blue()
a = color.alpha() # アルファ値 (透明度)

# RGB値の設定
color.setRed(200)
color.setGreen(100)
color.setBlue(50)
color.setAlpha(200) # 透明度設定
```

HSV値の操作

```
color = QColor()

# HSV値から設定
color.setHsv(120, 255, 255) # 色相, 彩度, 明度

# HSV値の取得
h = color.hue()
s = color.saturation()
v = color.value()
```

テーマとスタイルシート

ライトテーマとダークテーマ

```
def apply_light_theme(self):
    """ライトテーマの適用"""
    self.setStyleSheet("""
        QWidget {
            background-color: #ffffff;
            color: #2c3e50;
        }
        QGroupBox {
            background-color: #f8f9fa;
        }
    """)

def apply_dark_theme(self):
    """ダークテーマの適用"""
    self.setStyleSheet("""
        QWidget {
            background-color: #2c3e50;
            color: #ecf0f1;
        }
        QGroupBox {
            background-color: #34495e;
        }
    """)
```

スタイルシートでの色の使用

```
# ウィジェットのスタイル設定
widget.setStyleSheet("""
    QPushButton {
```

```

background-color: #3498db;
color: white;
border: 2px solid #2980b9;
padding: 10px;
border-radius: 5px;
font-weight: bold;
}
QPushButton:hover {
background-color: #2980b9;
}
""")
```

Material Design カラーパレット

推奨色の組み合わせ

```

# Primary Colors
RED_500 = QColor("#F44336")      # 赤
PINK_500 = QColor("#E91E63")      # ピンク
PURPLE_500 = QColor("#9C27B0")    # 紫
DEEP_PURPLE_500 = QColor("#673AB7") # 深紫

INDIGO_500 = QColor("#3F51B5")    # インディゴ
BLUE_500 = QColor("#2196F3")      # 青
LIGHT_BLUE_500 = QColor("#03A9F4") # 水色
CYAN_500 = QColor("#00BCD4")      # シアン

TEAL_500 = QColor("#009688")      # ティール
GREEN_500 = QColor("#4CAF50")      # 緑
LIGHT_GREEN_500 = QColor("#8BC34A") # 明るい緑
LIME_500 = QColor("#CDDC39")      # ライム

YELLOW_500 = QColor("#FFEB3B")      # 黄色
AMBER_500 = QColor("#FFC107")      # アンバー
ORANGE_500 = QColor("#FF9800")      # オレンジ
DEEP_ORANGE_500 = QColor("#FF5722") # 深いオレンジ
```

標準色名

色名	RGB値	16進数	日本語
red	(255, 0, 0)	#FF0000	赤
green	(0, 128, 0)	#008000	緑
blue	(0, 0, 255)	#0000FF	青
yellow	(255, 255, 0)	#FFFF00	黄色
cyan	(0, 255, 255)	#00FFFF	シアン
magenta	(255, 0, 255)	#FF00FF	マゼンタ
black	(0, 0, 0)	#000000	黒
white	(255, 255, 255)	#FFFFFF	白
gray	(128, 128, 128)	#808080	灰色
orange	(255, 165, 0)	#FFA500	オレンジ

サンプルアプリケーションの実行

各サンプルアプリケーションは独立して実行可能です：

```
# 基本的な色の作成  
python qcolor_basic.py  
  
# RGB値の制御  
python qcolor_rgb_control.py  
  
# カラーパレットとテーマ  
python qcolor_palette.py
```

実用的なTips

1. 適切なテキスト色の選択

```
def get_text_color(background_color):  
    """背景色に基づいて適切なテキスト色を返す"""  
    luminance = (0.299 * background_color.red() +  
                 0.587 * background_color.green() +  
                 0.114 * background_color.blue()) / 255  
    return "white" if luminance < 0.5 else "black"
```

2. 色の有効性チェック

```
color = QColor("#FF0000")  
if color.isValid():  
    print("有効な色です")  
else:  
    print("無効な色です")
```

3. 色の比較

```
color1 = QColor(255, 0, 0)  
color2 = QColor("#FF0000")  
  
if color1 == color2:  
    print("同じ色です")
```

参考リンク

- [Qt Documentation - QColor](#)
- [PySide6 QColor](#)

QSize, QRect, QPoint リファレンス

PySide6における基本図形クラス（座標、サイズ、矩形）についてのリファレンス資料です。

概要

これらのクラスは、GUI要素の位置、サイズ、領域を表現するための基本的な図形クラスです。

QPoint - 座標点

基本的な使用方法

```
from PySide6.QtCore import QPoint

# 座標点の作成
point1 = QPoint(10, 20)
point2 = QPoint(50, 100)

# 座標値の取得
x = point1.x()
y = point1.y()

# 座標値の設定
point1.setX(30)
point1.setY(40)
```

座標計算

```
p1 = QPoint(10, 20)
p2 = QPoint(30, 40)

# 座標の加算・減算
p3 = p1 + p2 # QPoint(40, 60)
p4 = p2 - p1 # QPoint(20, 20)

# 座標の判定
is_null = p1.isNull() # 座標が(0, 0)かどうか
```

QSize - サイズ

基本的な使用方法

```
from PySide6.QtCore import QSize

# サイズの作成
size1 = QSize(100, 50)
size2 = QSize(200, 150)

# サイズの取得
width = size1.width()
height = size1.height()

# サイズの設定
size1.setWidth(120)
size1.setHeight(80)
```

サイズ計算

```
size1 = QSize(100, 50)
size2 = QSize(200, 150)

# サイズの拡張・縮小
expanded = size1.expandedTo(size2) # より大きいサイズを取得
bounded = size1.boundedTo(size2) # より小さいサイズを取得

# 有効性チェック
is_valid = size1.isValid() # 幅・高さが正の値かどうか
```

QRect - 矩形

基本的な使用方法

```
from PySide6.QtCore import QRect, QPoint, QSize

# 矩形の作成方法
rect1 = QRect(10, 20, 100, 50) # x, y, width, height
rect2 = QRect(QPoint(10, 20), QSize(100, 50)) # 位置とサイズから

# 矩形の取得
x = rect1.x()
y = rect1.y()
width = rect1.width()
height = rect1.height()

# 角の座標取得
top_left = rect1.topLeft()
top_right = rect1.topRight()
bottom_left = rect1.bottomLeft()
bottom_right = rect1.bottomRight()
```

矩形の操作

```
rect = QRect(10, 20, 100, 50)

# 位置の移動
rect.moveTo(50, 100) # 絶対位置に移動
rect.moveTopLeft(QPoint(0, 0)) # 左上角を指定位置に移動

# サイズの変更
rect.setSize(QSize(150, 75))
rect.setWidth(200)
rect.setHeight(100)

# 矩形の拡張・縮小
rect.adjust(-5, -5, 10, 10) # 左上を(-5, -5)、右下を(10, 10)移動
```

矩形の判定

```
rect1 = QRect(10, 20, 100, 50)
rect2 = QRect(50, 40, 100, 50)
point = QPoint(60, 45)

# 点が矩形内にあるかチェック
contains_point = rect1.contains(point)

# 矩形同士の重なりチェック
intersects = rect1.intersects(rect2)
intersection = rect1.intersected(rect2) # 重なり部分の矩形
```

```
# 矩形の結合
united = rect1.united(rect2) # 両方を含む最小矩形
```

ウィジェットでの使用例

ウィジェットの位置・サイズ設定

```
from PySide6.QtWidgets import QWidget, QPushButton

widget = QWidget()
button = QPushButton("Click me", widget)

# ジオメトリ設定
widget.setGeometry(QRect(100, 100, 300, 200))
button.setGeometry(10, 10, 100, 30)

# 位置とサイズを個別に設定
widget.move(QPoint(150, 150))
widget.resize(QSize(400, 300))

# 現在の値を取得
current_pos = widget.pos()
current_size = widget.size()
current_rect = widget.geometry()
```

カスタム描画での使用

```
from PySide6.QtGui import QPainter
from PySide6.QtWidgets import QWidget

class CustomWidget(QWidget):
    def paintEvent(self, event):
        painter = QPainter(self)

        # 矩形の描画
        rect = QRect(10, 10, 100, 50)
        painter.drawRect(rect)

        # 円の描画（矩形に内接）
        circle_rect = QRect(150, 10, 50, 50)
        painter.drawEllipse(circle_rect)
```

よく使用されるサイズ定数

```
from PySide6.QtCore import QSize

# 標準的なサイズ
ICON_SIZE_SMALL = QSize(16, 16)
ICON_SIZE_MEDIUM = QSize(32, 32)
ICON_SIZE_LARGE = QSize(48, 48)

BUTTON_SIZE_SMALL = QSize(80, 25)
BUTTON_SIZE_MEDIUM = QSize(100, 30)
BUTTON_SIZE_LARGE = QSize(120, 35)

WINDOW_SIZE_SMALL = QSize(400, 300)
WINDOW_SIZE_MEDIUM = QSize(800, 600)
WINDOW_SIZE_LARGE = QSize(1200, 900)
```

参考リンク

- [Qt Documentation - QPoint](#)
- [Qt Documentation - QSize](#)

- [Qt Documentation - QRect](#)

QPixmap・QIcon リファレンス

QPixmapとQIconは、PySide6における画像とアイコンの処理を担当する重要なクラスです。

QPixmap

概要

QPixmapは画像データを保持し、表示に最適化されたクラスです。メモリ効率が良く、高速な描画が可能です。

基本的な作成方法

```
# ファイルから読み込み
pixmap = QPixmap("image.png")

# サイズを指定して作成
pixmap = QPixmap(100, 100)

# 空のピクセルマップ
pixmap = QPixmap()
```

よく使用されるメソッド

サイズ関連

- `width()` - 幅を取得
- `height()` - 高さを取得
- `size()` - QSizeとしてサイズを取得
- `scaled(width, height, mode)` - スケールした複製を作成

変形・変換

- `scaled()` - サイズ変更
- `transformed()` - 回転・反転等の変形
- `copy(x, y, width, height)` - 部分的にコピー

保存・読み込み

- `load(filename)` - ファイル読み込み
- `save(filename, format)` - ファイル保存
- `loadFromData(data)` - バイトデータから読み込み

状態確認

- `isNull()` - 空かどうかチェック
- `width()`, `height()` - サイズ取得

スケーリングモード

```
from PySide6.QtCore import Qt

# アスペクト比を保持して拡大縮小
pixmap.scaled(100, 100, Qt.KeepAspectRatio)
```

```
# アスペクト比を無視して拡大縮小  
pixmap.scaled(100, 100, Qt.IgnoreAspectRatio)  
  
# アスペクト比を保持し、はみ出る部分をクロップ  
pixmap.scaled(100, 100, Qt.KeepAspectRatioByExpanding)
```

変形の種類

```
from PySide6.QtGui import QTransform  
  
# 回転  
transform = QTransform()  
transform.rotate(45)  
rotated_pixmap = pixmap.transformed(transform)  
  
# 反転  
flipped_h = pixmap.transformed(QTransform().scale(-1, 1)) # 水平反転  
flipped_v = pixmap.transformed(QTransform().scale(1, -1)) # 垂直反転
```

QIcon

概要

QIconは複数のサイズ・状態のピクセルマップを管理し、UI要素で使用されるアイコンを表現します。

基本的な作成方法

```
# ファイルから作成  
icon = QIcon("icon.png")  
  
# QPixmapから作成  
icon = QIcon(pixmap)  
  
# 空のアイコン  
icon = QIcon()
```

モードと状態

```
from PySide6.QtGui import QIcon  
  
# モード  
QIcon.Normal # 通常状態  
QIcon.Disabled # 無効状態  
QIcon.Active # アクティブ状態 (ホバー等)  
QIcon.Selected # 選択状態  
  
# 状態  
QIcon.Off # オフ状態  
QIcon.On # オン状態
```

複数サイズの管理

```
icon = QIcon()  
  
# 異なるサイズを追加  
icon.addFile("icon_16.png", QSize(16, 16))  
icon.addFile("icon_32.png", QSize(32, 32))  
icon.addFile("icon_64.png", QSize(64, 64))  
  
# QPixmapを直接追加
```

```
icon.addPixmap(pixmap_16, QIcon.Normal, QIcon.Off)
icon.addPixmap(pixmap_32, QIcon.Normal, QIcon.Off)
```

よく使用されるメソッド

サイズ関連

- `availableSizes()` - 利用可能なサイズ一覧
- `actualSize(size)` - 指定サイズに最も近い実際のサイズ

ピクセルマップ取得

- `pixmap(size)` - 指定サイズのQPixmapを取得
- `pixmap(width, height)` - 指定サイズのQPixmapを取得

状態確認

- `isNull()` - 空かどうかチェック

実用的な使用例

画像の動的生成

```
# グラデーション画像の生成
pixmap = QPixmap(200, 100)
painter = QPainter(pixmap)
gradient = QLinearGradient(0, 0, 200, 0)
gradient.setColorAt(0, QColor(255, 0, 0))
gradient.setColorAt(1, QColor(0, 0, 255))
painter.fillRect(pixmap.rect(), gradient)
painter.end()
```

アイコンの状態管理

```
# ボタンの状態に応じたアイコン
button_icon = QIcon()
button_icon.addFile("play.png", QSize(), QIcon.Normal, QIcon.Off)
button_icon.addFile("pause.png", QSize(), QIcon.Normal, QIcon.On)
button.setIcon(button_icon)
button.setCheckable(True) # トグル可能にする
```

画像効果の適用

```
# セピア効果
def apply_sepia(pixmap):
    image = pixmap.toImage()
    for y in range(image.height()):
        for x in range(image.width()):
            color = QColor(image.pixel(x, y))
            gray = int(0.299 * color.red() + 0.587 * color.green() + 0.114 * color.blue())
            sepia_r = min(255, int(gray * 1.2))
            sepia_g = min(255, int(gray * 1.0))
            sepia_b = min(255, int(gray * 0.8))
            image.setPixel(x, y, QColor(sepia_r, sepia_g, sepia_b).rgb())
    return QPixmap.fromImage(image)
```

パフォーマンスの考慮事項

メモリ使用量

- QPixmapは表示用に最適化されているため、メモリ使用量に注意
- 大きな画像は必要に応じてスケールダウン
- 不要になったQPixmapは適切に解放

描画パフォーマンス

- QPixmapはGPUで高速描画可能
- 頻繁に変更される画像には QImageを使用検討
- キャッシュを活用して再描画を最小化

推奨事項

- アイコンには複数サイズを用意
- 高DPI対応として@2x画像も準備
- ファイル形式はPNG推奨（透明度対応）
- SVGアイコンも検討（スケーラブル）

実際のサンプルコード

[samples/q930_qpixmap_qicon/qpixmap_qicon_01.py](#) を参照してください。

関連クラス

- [QImage](#) - 画像データの直接操作用
- [QPainter](#) - 描画処理用
- [QBrush](#) - ブラシパターン用
- [QColor](#) - 色管理用

QStyleSheet リファレンス

PySide6におけるスタイルシート（CSS風スタイリング）についての包括的なリファレンス資料です。

概要

StyleSheetは、Qtウィジェットの外観をCSSライクな記法でカスタマイズできる機能です。

基本的な構文

セレクター

```
/* ウィジェットタイプセレクター */
QPushButton { }

/* クラス名セレクター */
QWidget { }

/* オブジェクト名セレクター */
QPushButton#myButton { }

/* 子セレクター */
QDialog QPushButton { }

/* 疑似状態セレクター */
QPushButton:hover { }
QPushButton:pressed { }
QPushButton:disabled { }
```

プロパティ

```
QPushButton {
    background-color: #3498db;
    color: white;
    border: 2px solid #2980b9;
    border-radius: 4px;
    padding: 8px 16px;
    font-size: 14px;
    font-weight: bold;
}
```

よく使用されるプロパティ

背景・前景

```
QWidget {
    background-color: #f0f0f0;
    background-image: url(background.png);
    background-repeat: no-repeat;
    background-position: center;
    color: #333333;
}
```

ボーダー

```
QLineEdit {  
    border: 2px solid gray;  
    border-radius: 5px;  
    border-style: solid;  
    border-top: 1px solid red;  
    border-right: 2px dashed blue;  
}
```

フォント

```
QLabel {  
    font-family: Arial, sans-serif;  
    font-size: 12pt;  
    font-weight: bold;  
    font-style: italic;  
    text-decoration: underline;  
}
```

マージン・パディング

```
QPushButton {  
    margin: 10px;  
    margin-top: 5px;  
    margin-right: 15px;  
    padding: 8px 16px;  
    padding-left: 20px;  
}
```

疑似状態

一般的な疑似状態

```
/* ホバー時 */  
QPushButton:hover {  
    background-color: #2980b9;  
}  
  
/* 押下時 */  
QPushButton:pressed {  
    background-color: #21618c;  
}  
  
/* 無効時 */  
QPushButton:disabled {  
    background-color: #bdc3c7;  
    color: #7f8c8d;  
}  
  
/* フォーカス時 */  
QLineEdit:focus {  
    border: 2px solid #3498db;  
}  
  
/* チェック時 (チェックボックス、ラジオボタン) */  
QCheckBox:checked {  
    color: #27ae60;  
}
```

ウィジェット固有の疑似状態

```
/* QTabWidget */
QTabWidget::tab-bar:selected {
    background-color: #3498db;
}

/* QScrollBar */
QScrollBar:vertical {
    background-color: #f0f0f0;
}

/* QComboBox */
QComboBox:drop-down {
    border: none;
}
```

ウィジェット別スタイリング例

QPushButton

```
QPushButton {
    background-color: #3498db;
    border: none;
    color: white;
    padding: 8px 16px;
    border-radius: 4px;
    font-weight: bold;
}

QPushButton:hover {
    background-color: #2980b9;
}

QPushButton:pressed {
    background-color: #21618c;
}
```

QLineEdit

```
QLineEdit {
    border: 2px solid #bdc3c7;
    border-radius: 4px;
    padding: 8px;
    font-size: 14px;
    background-color: white;
}

QLineEdit:focus {
    border-color: #3498db;
}

QLineEdit:disabled {
    background-color: #ecf0f1;
    color: #7f8c8d;
}
```

QTabWidget

```
QTabWidget::pane {
    border: 1px solid #bdc3c7;
    background-color: white;
}
```

```

QTabWidget::tab-bar {
    left: 5px;
}

QTabBar::tab {
    background-color: #ecf0f1;
    border: 1px solid #bdc3c7;
    padding: 8px 16px;
    margin-right: 2px;
}

QTabBar::tab:selected {
    background-color: white;
    border-bottom: none;
}

QTabBar::tab:hover {
    background-color: #d5dbdb;
}

```

QScrollBar

```

QScrollBar::vertical {
    background-color: #f8f9fa;
    width: 12px;
    border-radius: 6px;
}

QScrollBar::handle:vertical {
    background-color: #bdc3c7;
    border-radius: 6px;
    min-height: 20px;
}

QScrollBar::handle:vertical:hover {
    background-color: #95a5a6;
}

```

高度なテクニック

グラデーション

```

QPushButton {
    background: qlineargradient(
        x1: 0, y1: 0, x2: 0, y2: 1,
        stop: 0 #3498db,
        stop: 1 #2980b9
    );
}

/* 放射状グラデーション */
QWidget {
    background: qradialgradient(
        cx: 0.5, cy: 0.5, radius: 0.5,
        stop: 0 white,
        stop: 1 gray
    );
}

```

アニメーション対応

```

QPushButton {
    background-color: #3498db;
    transition: background-color 0.3s ease;
}

```

```
QPushButton:hover {  
    background-color: #2980b9;  
}
```

サブコントロール

```
QComboBox {  
    border: 1px solid gray;  
    border-radius: 3px;  
    padding: 1px 18px 1px 3px;  
}  
  
QComboBox::drop-down {  
    subcontrol-origin: padding;  
    subcontrol-position: top right;  
    width: 15px;  
    border-left: 1px solid darkgray;  
}  
  
QComboBox::down-arrow {  
    image: url(down_arrow.png);  
}
```

テーマの実装例

ダークテーマ

```
QWidget {  
    background-color: #2c3e50;  
    color: #ecf0f1;  
}  
  
QPushButton {  
    background-color: #34495e;  
    border: 1px solid #5d6d7e;  
    color: #ecf0f1;  
    padding: 8px 16px;  
    border-radius: 4px;  
}  
  
QPushButton:hover {  
    background-color: #5d6d7e;  
}  
  
QLineEdit {  
    background-color: #34495e;  
    border: 1px solid #5d6d7e;  
    color: #ecf0f1;  
    padding: 5px;  
    border-radius: 3px;  
}
```

マテリアルデザイン風

```
QPushButton {  
    background-color: #2196F3;  
    border: none;  
    color: white;  
    padding: 12px 24px;  
    border-radius: 4px;  
    font-weight: 500;  
    text-transform: uppercase;  
}
```

```
QPushButton:hover {
    background-color: #1976D2;
    box-shadow: 0 2px 4px rgba(0,0,0,0.2);
}

QPushButton:pressed {
    background-color: #0D47A1;
}
```

デバッグとツール

スタイルシートのデバッグ

```
# スタイルシートの適用確認
widget.setStyleSheet("border: 2px solid red;")

# 現在のスタイルシート取得
current_style = widget.styleSheet()
print(current_style)
```

動的スタイル変更

```
def toggle_theme(self):
    if self.dark_theme:
        self.setStyleSheet(self.light_theme_css)
    else:
        self.setStyleSheet(self.dark_theme_css)
    self.dark_theme = not self.dark_theme
```

注意点とベストプラクティス

1. **パフォーマンス**: 過度に複雑なスタイルシートは描画性能に影響する
2. **継承**: 親ウィジェットのスタイルが子に継承される
3. **特異性**: より具体的なセレクターが優先される
4. **プラットフォーム**: OS固有のスタイルを上書きする場合がある

参考リンク

- [Qt Documentation - Qt Style Sheets](#)
- [Qt Style Sheets Reference](#)

Signal & Slot リファレンス

PySide6におけるシグナル・スロット機能についての詳細なリファレンス資料です。

概要

シグナル・スロットは、Qtのコア機能の一つで、オブジェクト間の安全で柔軟な通信を提供します。

基本概念

シグナル (Signal)

- イベント発生時に送出される
- 引数を持つことができる
- 複数のスロットに接続可能

スロット (Slot)

- シグナルを受信する関数・メソッド
- 通常のPython関数として定義可能
- 戻り値は無視される

基本的な使用方法

シグナルの定義

```
from PySide6.QtCore import QObject, Signal

class MyClass(QObject):
    # 引数なしシグナル
    finished = Signal()

    # 引数ありシグナル
    dataChanged = Signal(str)
    valueUpdated = Signal(int, str)

    # オーバーロードシグナル
    progress = Signal(int)
    progress = Signal(str)
```

シグナルとスロットの接続

```
from PySide6.QtWidgets import QPushButton

# 基本的な接続
button = QPushButton("Click me")
button.clicked.connect(my_function)

# メソッドとの接続
button.clicked.connect(self.handle_click)

# ラムダ関数との接続
button.clicked.connect(lambda: print("Button clicked!"))
```

```
# 引数付きシグナルの接続  
lineEdit.textChanged.connect(self.onTextChanged)
```

接続の解除

```
# 特定の接続を解除  
button.clicked.disconnect(my_function)  
  
# すべての接続を解除  
button.clicked.disconnect()  
  
# オブジェクトからのすべての接続を解除  
button.disconnect()
```

実用的な例

カスタムシグナルの使用

```
from PySide6.QtCore import QObject, Signal  
from PySide6.QtWidgets import QWidget, QPushButton, QVBoxLayout  
  
class Counter(QObject):  
    """カウンター値の変更を通知するクラス"""  
  
    # シグナルの定義  
    valueChanged = Signal(int)  
    limitReached = Signal()  
  
    def __init__(self):  
        super().__init__()  
        self._value = 0  
        self._limit = 10  
  
    def increment(self):  
        self._value += 1  
        self.valueChanged.emit(self._value) # シグナル送出  
  
        if self._value >= self._limit:  
            self.limitReached.emit()  
  
    def reset(self):  
        self._value = 0  
        self.valueChanged.emit(self._value)  
  
class CounterWidget(QWidget):  
    def __init__(self):  
        super().__init__()  
        self.setupUi()  
        self.setupConnections()  
  
    def setupUi(self):  
        layout = QVBoxLayout()  
        self.increment_btn = QPushButton("増加")  
        self.reset_btn = QPushButton("リセット")  
        layout.addWidget(self.increment_btn)  
        layout.addWidget(self.reset_btn)  
        self.setLayout(layout)  
  
        self.counter = Counter()  
  
    def setupConnections(self):  
        # ボタンのクリックイベントとカウンターを接続  
        self.increment_btn.clicked.connect(self.counter.increment)  
        self.reset_btn.clicked.connect(self.counter.reset)  
  
        # カウンターの変更イベントとUIの更新を接続  
        self.counter.valueChanged.connect(self.updateDisplay)
```

```

    self.counter.limitReached.connect(self.handle_limit_reached)

def update_display(self, value):
    self.setWindowTitle(f"カウンター: {value}")

def handle_limit_reached(self):
    print("制限値に達しました!")

```

データバインディングの実装

```

from PySide6.QtCore import QObject, Signal, Property

class Model(QObject):
    """データモデルクラス"""

    nameChanged = Signal(str)
    ageChanged = Signal(int)

    def __init__(self):
        super().__init__()
        self._name = ""
        self._age = 0

    @Property(str, notify=nameChanged)
    def name(self):
        return self._name

    @name.setter
    def name(self, value):
        if self._name != value:
            self._name = value
            self.nameChanged.emit(value)

    @Property(int, notify=ageChanged)
    def age(self):
        return self._age

    @age.setter
    def age(self, value):
        if self._age != value:
            self._age = value
            self.ageChanged.emit(value)

class View(QWidget):
    """ビュークラス"""

    def __init__(self, model):
        super().__init__()
        self.model = model
        self.setup_ui()
        self.setup_bindings()

    def setup_bindings(self):
        # モデルの変更をビューに反映
        self.model.nameChanged.connect(self.name_edit.setText)
        self.model.ageChanged.connect(self.age_spinbox.setValue)

        # ビューの変更をモデルに反映
        self.name_edit.textChanged.connect(self.model.name.fset)
        self.age_spinbox.valueChanged.connect(self.model.age.fset)

```

高度な機能

シグナルのブロック

```

# シグナルを一時的にブロック
button.blockSignals(True)

```

```

button.setText("新しいテキスト") # textChangedシグナルが発生しない
button.blockSignals(False)

# コンテキストマネージャーとして使用
from contextlib import contextmanager

@contextmanager
def blocked_signals(obj):
    old_state = obj.blockSignals(True)
    try:
        yield obj
    finally:
        obj.blockSignals(old_state)

# 使用例
with blocked_signals(button):
    button.setText("一時的なテキスト")

```

カスタム接続タイプ

```

from PySide6.QtCore import Qt

# 異なる接続タイプ
button.clicked.connect(slot, Qt.ConnectionType.DirectConnection)
button.clicked.connect(slot, Qt.ConnectionType.QueuedConnection)
button.clicked.connect(slot, Qt.ConnectionType.AutoConnection)

```

シグナルチェーン

```

class SignalChain(QObject):
    step1Completed = Signal()
    step2Completed = Signal()
    allCompleted = Signal()

    def __init__(self):
        super().__init__()
        # シグナルをチェーン接続
        self.step1Completed.connect(self.start_step2)
        self.step2Completed.connect(self.all_completed)

    def start_process(self):
        # 処理開始
        self.step1Completed.emit()

    def start_step2(self):
        # ステップ2実行
        self.step2Completed.emit()

    def all_completed(self):
        self.allCompleted.emit()

```

デバッグとトラブルシューティング

接続状態の確認

```

from PySide6.QtCore import QObject

# 接続数の確認
connections = QObject.receivers(button.clicked)
print(f"接続数: {connections}")

# デバッグ情報の出力
import os
os.environ['QT_LOGGING_RULES'] = 'qt.qpa.debug=true'

```

よくある問題と解決策

```
# 1. オブジェクトの生存期間の問題
class BadExample:
    def __init__(self):
        button = QPushButton()
        button.clicked.connect(self.handle_click)
        # buttonがスコープを出ると削除される可能性

class GoodExample:
    def __init__(self):
        self.button = QPushButton() # インスタンス変数として保持
        self.button.clicked.connect(self.handle_click)

# 2. 循環参照の回避
class Parent(QObject):
    def __init__(self):
        super().__init__()
        self.child = Child(self)

class Child(QObject):
    def __init__(self, parent):
        super().__init__(parent) # 親を設定
        # 親への参照は弱参照として扱われる
```

パフォーマンス最適化

効率的なシグナル使用

```
# 頻繁に発生するシグナルの最適化
from PySide6.QtCore import QTimer

class OptimizedSignal(QObject):
    dataChanged = Signal()

    def __init__(self):
        super().__init__()
        self._pending_update = False
        self._timer = QTimer()
        self._timer.timeout.connect(self._emit_delayed)
        self._timer.setSingleShot(True)

    def request_update(self):
        if not self._pending_update:
            self._pending_update = True
            self._timer.start(100) # 100ms後に実際のシグナルを送出

    def _emit_delayed(self):
        self._pending_update = False
        self.dataChanged.emit()
```

ベストプラクティス

1. **命名規則:** シグナル名は動詞形 (clicked, finished等) を使用
2. **引数の設計:** 必要最小限の情報のみ渡す
3. **接続の管理:** 適切なタイミングで disconnect を呼ぶ
4. **エラーハンドリング:** スロット内でのエラーハンドリングを適切に行う

```
class BestPracticeExample(QObject):
    # 適切なシグナル名
    fileLoaded = Signal(str) # ファイルパスを渡す
    progressChanged = Signal(int) # 進行率 (0-100)

    def __init__(self):
        super().__init__()

    def safe_slot(self, data):
        """安全なスロットの実装例"""
        try:
            # 処理内容
            self.process_data(data)
        except Exception as e:
            print(f"エラーが発生しました: {e}")
            # エラー用シグナルを送出することも可能
```

参考リンク

- [Qt Documentation - Signals & Slots](#)
- [PySide6 Signals & Slots](#)

Qt定数とフラグ

Qtクラスには、PySide6で使用される様々な定数とフラグが定義されています。これらは配置、マウスボタン、キーボードなどの設定に使用されます。

インポート

```
from PySide6.QtCore import Qt
```

アライメント（配置）フラグ

水平配置

定数	説明
Qt.AlignmentFlag.AlignLeft	左揃え
Qt.AlignmentFlag.AlignRight	右揃え
Qt.AlignmentFlag.AlignHCenter	水平中央揃え
Qt.AlignmentFlag.AlignJustify	両端揃え

垂直配置

定数	説明
Qt.AlignmentFlag.AlignTop	上揃え
Qt.AlignmentFlag.AlignBottom	下揃え
Qt.AlignmentFlag.AlignVCenter	垂直中央揃え

組み合わせ配置

定数	説明
Qt.AlignmentFlag.AlignCenter	中央揃え（水平+垂直）

使用例

```
from PySide6.QtWidgets import QLabel
from PySide6.QtCore import Qt

label = QLabel("中央揃えのテキスト")
label.setAlignment(Qt.AlignmentFlag.AlignCenter)

# 複数のフラグを組み合わせ
label.setAlignment(Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignTop)
```

オリエンテーション（方向）

定数	説明
Qt.Orientation.Horizontal	水平方向
Qt.Orientation.Vertical	垂直方向

使用例

```
from PySide6.QtWidgets import QSlider
from PySide6.QtCore import Qt

horizontal_slider = QSlider(Qt.Orientation.Horizontal)
vertical_slider = QSlider(Qt.Orientation.Vertical)
```

マウスボタン

定数	説明
Qt.MouseButton.NoButton	マウスボタンなし
Qt.MouseButton.LeftButton	左ボタン
Qt.MouseButton.RightButton	右ボタン
Qt.MouseButton.MiddleButton	中ボタン

使用例

```
def mousePressEvent(self, event):
    if event.button() == Qt.MouseButton.LeftButton:
        print("左クリック")
    elif event.button() == Qt.MouseButton.RightButton:
        print("右クリック")
```

キーボード修飾キー

定数	説明
Qt.KeyboardModifier.NoModifier	修飾キーなし
Qt.KeyboardModifier.ShiftModifier	Shiftキー
Qt.KeyboardModifier.ControlModifier	Ctrlキー
Qt.KeyboardModifier.AltModifier	Altキー
Qt.KeyboardModifier.MetaModifier	Metaキー（WindowsキーまたはCmdキー）

使用例

```
def keyPressEvent(self, event):
    if event.modifiers() & Qt.KeyboardModifier.ControlModifier:
        if event.key() == Qt.Key.Key_S:
            print("Ctrl+Sが押されました")
```

よく使用されるキーコード

制御キー

定数	説明
Qt.Key.Key_Enter	Enterキー
Qt.Key.Key_Return	Returnキー
Qt.Key.Key_Escape	Escapeキー
Qt.Key.Key_Space	スペースキー
Qt.Key.Key_Tab	Tabキー
Qt.Key.Key_Backspace	Backspaceキー
Qt.Key.Key_Delete	Deleteキー

矢印キー

定数	説明
Qt.Key.Key_Up	上矢印
Qt.Key.Key_Down	下矢印
Qt.Key.Key_Left	左矢印
Qt.Key.Key_Right	右矢印

文字キー（例）

定数	説明
Qt.Key.Key_A	Aキー
Qt.Key.Key_B	Bキー
...	(他のアルファベット)
Qt.Key.Key_Z	Zキー

数字キー

定数	説明
Qt.Key.Key_0	0キー
Qt.Key.Key_1	1キー
...	(他の数字)
Qt.Key.Key_9	9キー

ウィンドウフラグ

基本ウィンドウタイプ

定数	説明
Qt.WindowType.Widget	標準ウィジエット
Qt.WindowType.Window	ウィンドウ
Qt.WindowType.Dialog	ダイアログ
Qt.WindowType.Sheet	シート (macOS)
Qt.WindowType.Drawer	ドロワー (macOS)
Qt.WindowType.Popup	ポップアップ
Qt.WindowType.Tool	ツールウィンドウ
Qt.WindowType.ToolTip	ツールチップ
Qt.WindowType.SplashScreen	スプラッシュスクリーン

ウィンドウヒント

定数	説明
Qt.WindowType.FramelessWindowHint	フレームなし
Qt.WindowType.WindowTitleHint	タイトルバー表示
Qt.WindowType.WindowSystemMenuHint	システムメニュー
Qt.WindowType.WindowMinimizeButtonHint	最小化ボタン
Qt.WindowType.WindowMaximizeButtonHint	最大化ボタン
Qt.WindowType.WindowCloseButtonHint	閉じるボタン

定数	説明
Qt.WindowType.WindowStaysOnTopHint	常に最前面

使用例

```
from PySide6.QtWidgets import QWidget
from PySide6.QtCore import Qt

# 常に最前面のフレームなしウィンドウ
widget = QWidget()
widget.setWindowFlags(
    Qt.WindowType.FramelessWindowHint |
    Qt.WindowType.WindowStaysOnTopHint
)
```

フォーカスポリシー

定数	説明
Qt.FocusPolicy.NoFocus	フォーカスなし
Qt.FocusPolicy.TabFocus	Tabキーでフォーカス
Qt.FocusPolicy.ClickFocus	クリックでフォーカス
Qt.FocusPolicy.StrongFocus	Tab + クリック
Qt.FocusPolicy.WheelFocus	マウスホイールでもフォーカス

使用例

```
from PySide6.QtWidgets import QPushButton
from PySide6.QtCore import Qt

button = QPushButton("ボタン")
button.setFocusPolicy(Qt.FocusPolicy.NoFocus) # フォーカスを受け取らない
```

サイズポリシー

定数	説明
QtSizePolicy.Policy.Fixed	固定サイズ
QtSizePolicy.Policy.Minimum	最小サイズ
QtSizePolicy.Policy.Maximum	最大サイズ
QtSizePolicy.Policy.Preferred	推奨サイズ
QtSizePolicy.Policy.Expanding	拡張可能
QtSizePolicy.Policy.MinimumExpanding	最小拡張
QtSizePolicy.Policy.Ignored	サイズヒント無視

カーソル形状

定数	説明
Qt.CursorShape.ArrowCursor	標準矢印
Qt.CursorShape.CrossCursor	十字
Qt.CursorShape.WaitCursor	待機（砂時計など）
Qt.CursorShape.IBeamCursor	テキスト入力（Iビーム）
Qt.CursorShape.SizeVerCursor	垂直リサイズ
Qt.CursorShape.SizeHorCursor	水平リサイズ

定数	説明
Qt.CursorShape.SizeBDiagCursor	斜めリサイズ (\)
Qt.CursorShape.SizeFDiagCursor	斜めリサイズ (/)
Qt.CursorShape.SizeAllCursor	全方向リサイズ
Qt.CursorShape.BankCursor	透明カーソル
Qt.CursorShape.SplitVCursor	垂直分割
Qt.CursorShape.SplitHCursor	水平分割
Qt.CursorShape.PointingHandCursor	指差し（リンク）
Qt.CursorShape.ForbiddenCursor	禁止
Qt.CursorShape.WhatsThisCursor	ヘルプ
Qt.CursorShape.BusyCursor	ビジー状態
Qt.CursorShape.OpenHandCursor	開いた手
Qt.CursorShape.ClosedHandCursor	閉じた手

使用例

```
from PySide6.QtWidgets import QLabel
from PySide6.QtCore import Qt

label = QLabel("ホバーしてください")
label.setCursor(Qt.CursorShape.PointingHandCursor)
```

実用的な使用例

アライメントの組み合わせ

```
from PySide6.QtWidgets import QLabel, QVBoxLayout, QWidget
from PySide6.QtCore import Qt

class AlignmentExample(QWidget):
    def __init__(self):
        super().__init__()
        layout = QVBoxLayout()
        self.setLayout(layout)

        alignments = [
            ("左上", Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignTop),
            ("中央上", Qt.AlignmentFlag.AlignHCenter | Qt.AlignmentFlag.AlignTop),
            ("右上", Qt.AlignmentFlag.AlignRight | Qt.AlignmentFlag.AlignTop),
            ("中央", Qt.AlignmentFlag.AlignCenter),
            ("左下", Qt.AlignmentFlag.AlignLeft | Qt.AlignmentFlag.AlignBottom),
        ]
        for text, alignment in alignments:
            label = QLabel(text)
            label.setAlignment(alignment)
            label.setStyleSheet("border: 1px solid black; min-height: 50px;")
            layout.addWidget(label)
```

キーボードショートカット

```
from PySide6.QtWidgets import QWidget
from PySide6.QtCore import Qt

class ShortcutExample(QWidget):
    def keyPressEvent(self, event):
        key = event.key()
        modifiers = event.modifiers()
```

```
if modifiers & Qt.KeyboardModifier.ControlModifier:
    if key == Qt.Key.Key_N:
        print("新規作成 (Ctrl+N)")
    elif key == Qt.Key.Key_O:
        print("開< (Ctrl+O)")
    elif key == Qt.Key.Key_S:
        print("保存 (Ctrl+S)")

    elif key == Qt.Key.Key_F1:
        print("ヘルプ (F1)")
    elif key == Qt.Key.Key_Escape:
        print("キャンセル (Esc)")

super().keyPressEvent(event)
```

注意事項

- フラグを組み合わせる場合は | 演算子を使用してください
- 古いPySide6バージョンでは一部の定数名が異なる場合があります
- Qt定数は大文字小文字を区別します
- 適切な名前空間 (`Qt.AlignmentFlag.AlignCenter` など) を使用してください