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 = QHBoxLayout()
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