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

スペーシングとマージン

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

ストレッチファクター

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

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("×")
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

# ウィジェ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 - QHBoxLayout
- Qt公式ドキュメント - Layout Management
- PySide6公式ドキュメント