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

setRowCount(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