tkinter.Toplevel リファレンス

概要

Toplevelウィジェットは、メインウィンドウとは別の新しいウィンドウを作成するためのウィジェットです。ダイアログボックス、設定ウィンドウ、サブウィンドウなどの作成に使用され、モーダル（他の操作をブロック）または非モーダル（並行操作可能）として動作させることができます。

基本的な使用方法

```
 import tkinter as tk

root = tk.Tk()
root.title("メインウィンドウ")

def open_sub_window():
    sub_window = tk.Toplevel(root)  # 親ウィンドウを指定
    sub_window.title("サブウィンドウ")
    sub_window.geometry("300x200")

    tk.Label(sub_window, text="これはサブウィンドウです").pack(pady=20)
    tk.Button(sub_window, text="閉じる", command=sub_window.destroy).pack()

tk.Button(root, text="サブウィンドウを開く", command=open_sub_window).pack(pady=20)

root.mainloop()
```

親ウィンドウの指定について

ベストプラクティス（推奨）

```
# 親ウィンドウを明示的に指定
sub_window = tk.Toplevel(parent)
```

アンチパターン（非推奨）

```
# 親ウィンドウを指定しない
sub_window = tk.Toplevel()
```

親ウィンドウ指定の有無による違い

項目	親指定あり	親指定なし
ウィンドウ階層	親ウィンドウに属する	独立したトップレベルウィンドウ
最小化連動	親と連動して最小化	独立して最小化
タスクバー表示	親のグループに含まれる	個別にタスクバーに表示
親ウィンドウ終了時	自動的に閉じられる	残存する可能性がある（ゾンビプロセス）
Z-order管理	親より前面に表示	システム依存

リスクレベル

- **Critical**: 親ウィンドウ終了時にサブウィンドウが残存（メモリリーク、ゾンビプロセス）
- **Major**: ユーザビリティ低下（タスクバーの混乱、ウィンドウ管理の困難）
- **Minor**: 一貫性のないウィンドウ動作

モーダル・非モーダルの使い分け

モーダルダイアログ（他の操作をブロック）

使用場面： - ユーザーからの入力が必須の場合 - 設定変更の確認が必要な場合
- エラーメッセージの表示 - ファイル保存前の未保存データ警告 - データ削除の確認

実装方法：

```
dialog = SomeDialog(parent)
dialog.transient(parent)  # 親に関連付け
dialog.grab_set()         # 他の操作をブロック
parent.wait_window(dialog)  # ダイアログが閉じるまで待機
```

非モーダルウィンドウ（並行操作可能）

使用場面： - ツールパレット、プロパティパネル - ログ表示ウィンドウ - リアルタイム情報表示 - 補助的な操作ウィンドウ - マルチタスクが必要な作業

実装方法：

```
window = SomeWindow(parent)
window.transient(parent)  # 親に関連付けるが、grab_setは呼ばない
```

主要なオプション

オプション	説明	デフォルト値	例
`master`	親ウィンドウ	None	`master=root`
`class_`	ウィンドウクラス名	'Toplevel'	`class_='Dialog'`
`bg`	背景色	SystemButtonFace	`bg='white'`
`width`	ウィンドウの幅	-	`width=400`
`height`	ウィンドウの高さ	-	`height=300`
`relief`	境界線のスタイル	'flat'	`relief='raised'`
`borderwidth`	境界線の幅	0	`borderwidth=2`

主要なメソッド

メソッド	説明	例
`geometry(geometry)`	ウィンドウサイズと位置を設定	`geometry('400x300+100+50')`
`title(string)`	ウィンドウタイトルを設定	`title('設定ダイアログ')`
`grab_set()`	モーダルにする（他の操作をブロック）	`grab_set()`
`grab_release()`	モーダルを解除	`grab_release()`
`transient(master)`	親ウィンドウに関連付け	`transient(root)`
`focus_set()`	フォーカスを設定	`focus_set()`
`iconify()`	ウィンドウを最小化	`iconify()`
`deiconify()`	最小化を解除	`deiconify()`
`withdraw()`	ウィンドウを非表示	`withdraw()`
`destroy()`	ウィンドウを閉じる	`destroy()`
`protocol(name, func)`	ウィンドウプロトコルを設定	`protocol('WM_DELETE_WINDOW', on_close)`

重複開防止のベストプラクティス

```
class MainApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.dialog = None  # ダイアログ参照を保持

    def open_dialog(self):
        # 重複開防止
        if self.dialog and self.dialog.winfo_exists():
            self.dialog.lift()  # 既存ダイアログを前面に
            return

        self.dialog = SomeDialog(self)
```

実用的な例

1. シンプルなモーダルダイアログ

```
import tkinter as tk
from tkinter import messagebox
```

```
class SimpleModalApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("モーダルダイアログテスト")
        self.geometry("300x200")
        self.current_dialog = None

        tk.Button(self, text="ダイアログを開く", command=self.open_dialog).pack(pady=20)
        tk.Button(self, text="テストボタン", command=self.test_click).pack()

    def open_dialog(self):
        if self.current_dialog and self.current_dialog.winfo_exists():
            self.current_dialog.lift()
            return

        dialog = SimpleDialog(self)
        self.current_dialog = dialog
        self.wait_window(dialog)  # ここで処理が停止
        self.current_dialog = None

    def test_click(self):
        messagebox.showinfo("テスト", "ダイアログが開いている間はクリックできません")

class SimpleDialog(tk.Toplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.title("モーダルダイアログ")
        self.geometry("200x100")

        self.transient(parent)  # 重要：親に関連付け
        self.grab_set()         # 重要：他の操作をブロック

        tk.Label(self, text="モーダルダイアログです").pack(pady=10)
        tk.Button(self, text="閉じる", command=self.destroy).pack()

if __name__ == "__main__":
    app = SimpleModalApp()
    app.mainloop()
```

2. シンプルな非モーダルウィンドウ

```
import tkinter as tk

class SimpleNonModalApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("非モーダルウィンドウテスト")
        self.geometry("300x200")
        self.sub_window = None

        button_frame = tk.Frame(self)
        button_frame.pack(pady=20)

        tk.Button(button_frame, text="サブウィンドウを開く",
                  command=self.open_sub_window).pack(side=tk.LEFT, padx=5)
        tk.Button(button_frame, text="閉じる",
                  command=self.close_sub_window).pack(side=tk.LEFT, padx=5)

        tk.Button(self, text="テストボタン", command=self.test_click).pack(pady=10)

    def open_sub_window(self):
        if self.sub_window and self.sub_window.winfo_exists():
            self.sub_window.lift()
            return

        self.sub_window = SubWindow(self)

    def close_sub_window(self):
        if self.sub_window and self.sub_window.winfo_exists():
            self.sub_window.destroy()
        self.sub_window = None

    def test_click(self):
        print("サブウィンドウが開いていてもクリックできます")

class SubWindow(tk.Toplevel):
    def __init__(self, parent):
        super().__init__(parent)
        self.parent = parent
        self.title("非モーダルウィンドウ")
        self.geometry("200x150")

        self.transient(parent)  # 親に関連付け（grab_setは呼ばない）

        tk.Label(self, text="非モーダルウィンドウです").pack(pady=10)

        self.counter = 0
        self.label = tk.Label(self, text=f"カウンター: {self.counter}")
        self.label.pack()

        tk.Button(self, text="+1", command=self.increment).pack(pady=5)
        tk.Button(self, text="閉じる", command=self.on_close).pack()

        self.protocol("WM_DELETE_WINDOW", self.on_close)

    def increment(self):
        self.counter += 1
```

```
        self.label.config(text=f"カウンター: {self.counter}")

    def on_close(self):
        self.parent.sub_window = None
        self.destroy()

if __name__ == "__main__":
    app = SimpleNonModalApp()
    app.mainloop()
```

注意点

- **親ウィンドウの指定**: 必ず親ウィンドウを指定してメモリリークを防ぐ
- **重複開防止**: 同じダイアログの重複開を適切に制御する
- **モーダルの適切な使用**: ユーザーの応答が必須の場合のみモーダルを使用
- **リソース管理**: 不要になったウィンドウは確実に `destroy()` する
- **フォーカス管理**: 適切な要素にフォーカスを設定してキーボード操作を改善