t920_keybinding.md - キーバインディング

`tkinter` では、キーボードショートカットやキーイベントをコントロールにバインドすることができます。これにより、特定のキーでボタンを押したような動作をさせたり、フォーカスを移動したりすることが可能です。

基本的なキーバインディング

bind() メソッド

すべての `tkinter` ウィジェットは `bind()` メソッドを持っており、キーイベントやマウスイベントを関数にバインドできます。

```
widget.bind('<KeyPattern>', callback_function)
```

キーパターンの書式

パターン	説明	例
`<Key>`	任意のキー	`<Key>`
`<Return>`	Enterキー	`<Return>`
`<Escape>`	Escapeキー	`<Escape>`
`<Tab>`	Tabキー	`<Tab>`
`<BackSpace>`	Backspaceキー	`<BackSpace>`
`<Delete>`	Deleteキー	`<Delete>`
`<F1>` - `<F12>`	ファンクションキー	`<F1>`, `<F5>`
`<Up>`, `<Down>`, `<Left>`, `<Right>`	矢印キー	`<Up>`
`<Control-c>`	Ctrl+C	`<Control-c>`
`<Alt-f>`	Alt+F	`<Alt-f>`
`<Shift-Tab>`	Shift+Tab	`<Shift-Tab>`
`<Control-Alt-d>`	Ctrl+Alt+D	`<Control-Alt-d>`

ショートカットでボタンを実行

基本例

```
import tkinter as tk

class App:
    def __init__(self, root):
        self.root = root

        # ボタンを作成
        self.button = tk.Button(root, text="クリックされました", command=self.button_click)
        self.button.pack(pady=10)

        # F1キーでボタンをトリガー
        root.bind('<F1>', lambda e: self.button_click())
        # Ctrl+Enterでもトリガー
        root.bind('<Control-Return>', lambda e: self.button_click())

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

root = tk.Tk()
app = App(root)
root.mainloop()
```

invoke() メソッドを使用

```
# ボタンの invoke() メソッドを直接呼び出す
root.bind('<F1>', lambda e: self.button.invoke())
```

フォーカス移動

focus_set() でフォーカス移動

```python
import tkinter as tk

class FocusApp:
    def __init__(self, root):
        self.root = root

        # 複数のエントリウィジェット
        self.entry1 = tk.Entry(root)
        self.entry1.pack(pady=5)

        self.entry2 = tk.Entry(root)
        self.entry2.pack(pady=5)

        self.entry3 = tk.Entry(root)
        self.entry3.pack(pady=5)

        # キーバインディング
        root.bind('<F2>', lambda e: self.entry1.focus_set())
        root.bind('<F3>', lambda e: self.entry2.focus_set())
        root.bind('<F4>', lambda e: self.entry3.focus_set())

        # Tabキーで次のウィジェットへ（デフォルト動作を補完）
        root.bind('<Control-Tab>', self.focus_next)
        root.bind('<Control-Shift-Tab>', self.focus_prev)

    def focus_next(self, event):
        event.widget.tk_focusNext().focus_set()

    def focus_prev(self, event):
        event.widget.tk_focusPrev().focus_set()
```

高度なキーバインディング

グローバルキーバインディング

```python
import tkinter as tk

class GlobalKeyApp:
    def __init__(self, root):
        self.root = root

        # メニューバー
        menubar = tk.Menu(root)
        root.config(menu=menubar)

        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="ファイル", menu=file_menu)
        file_menu.add_command(label="新規", command=self.new_file, accelerator="Ctrl+N")
        file_menu.add_command(label="開く", command=self.open_file, accelerator="Ctrl+O")
        file_menu.add_command(label="保存", command=self.save_file, accelerator="Ctrl+S")

        # テキストエリア
        self.text_area = tk.Text(root)
        self.text_area.pack(fill=tk.BOTH, expand=True)

        # グローバルキーバインディング
        self.bind_global_keys()

    def bind_global_keys(self):
        # ウィンドウ全体に対してバインド
        self.root.bind('<Control-n>', lambda e: self.new_file())
        self.root.bind('<Control-o>', lambda e: self.open_file())
        self.root.bind('<Control-s>', lambda e: self.save_file())
        self.root.bind('<Control-q>', lambda e: self.root.quit())

        # カスタムショートカット
        self.root.bind('<Control-d>', lambda e: self.duplicate_line())
        self.root.bind('<Control-slash>', lambda e: self.toggle_comment())

    def new_file(self):
        self.text_area.delete(1.0, tk.END)
        print("新しいファイル")

    def open_file(self):
        print("ファイルを開く")

    def save_file(self):
        print("ファイルを保存")

    def duplicate_line(self):
        # 現在の行を複製
        current_line = self.text_area.index(tk.INSERT).split('.')[0]
        line_content = self.text_area.get(f"{current_line}.0", f"{current_line}.end")
        self.text_area.insert(f"{current_line}.end", f"\n{line_content}")

    def toggle_comment(self):
        print("コメントの切り替え")
```

動的なキーバインディング

```python
import tkinter as tk
```

```
class DynamicKeyApp:
    def __init__(self, root):
        self.root = root

        self.label = tk.Label(root, text="キーを押してください", font=("Arial", 14))
        self.label.pack(pady=20)

        self.binding_enabled = True

        # 動的バインディング
        self.setup_number_keys()

        # バインディングの切り替えボタン
        self.toggle_btn = tk.Button(root, text="バインディング切り替え",
                                    command=self.toggle_binding)
        self.toggle_btn.pack(pady=10)

    def setup_number_keys(self):
        # 数字キー 1-9 にバインド
        for i in range(1, 10):
            self.root.bind(f'<Key-{i}>', lambda e, num=i: self.number_pressed(num))

    def number_pressed(self, number):
        if self.binding_enabled:
            self.label.config(text=f"数字 {number} が押されました")

    def toggle_binding(self):
        self.binding_enabled = not self.binding_enabled
        status = "有効" if self.binding_enabled else "無効"
        self.toggle_btn.config(text=f"バインディング: {status}")
```

コンテキスト固有のキーバインディング

ウィジェット固有のバインディング

```
 import tkinter as tk
from tkinter import ttk

class ContextKeyApp:
    def __init__(self, root):
        self.root = root

        # ノートブック（タブ）
        notebook = ttk.Notebook(root)
        notebook.pack(fill=tk.BOTH, expand=True)

        # テキストタブ
        text_frame = tk.Frame(notebook)
        notebook.add(text_frame, text="テキスト")

        self.text_widget = tk.Text(text_frame)
        self.text_widget.pack(fill=tk.BOTH, expand=True)

        # リストタブ
        list_frame = tk.Frame(notebook)
        notebook.add(list_frame, text="リスト")

        self.listbox = tk.Listbox(list_frame)
        self.listbox.pack(fill=tk.BOTH, expand=True)
        for i in range(20):
            self.listbox.insert(tk.END, f"アイテム {i+1}")

        # コンテキスト固有のバインディング
        self.setup_context_bindings()

    def setup_context_bindings(self):
        # テキストウィジェット固有
        self.text_widget.bind('<Control-d>', self.duplicate_text_line)
        self.text_widget.bind('<Control-k>', self.delete_text_line)

        # リストボックス固有
        self.listbox.bind('<Delete>', self.delete_list_item)
        self.listbox.bind('<Control-a>', self.select_all_items)

    def duplicate_text_line(self, event):
        widget = event.widget
        current_line = widget.index(tk.INSERT).split('.')[0]
        line_content = widget.get(f"{current_line}.0", f"{current_line}.end")
        widget.insert(f"{current_line}.end", f"\n{line_content}")

    def delete_text_line(self, event):
        widget = event.widget
        current_line = widget.index(tk.INSERT).split('.')[0]
        widget.delete(f"{current_line}.0", f"{int(current_line)+1}.0")

    def delete_list_item(self, event):
        selection = self.listbox.curselection()
        if selection:
            self.listbox.delete(selection[0])

    def select_all_items(self, event):
        self.listbox.select_set(0, tk.END)
```

アクセラレータキー（メニュー用）

accelerator オプション

```
 import tkinter as tk

class AcceleratorApp:
    def __init__(self, root):
        self.root = root

        # メニューバー
        menubar = tk.Menu(root)
        root.config(menu=menubar)

        # ファイルメニュー
        file_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="ファイル", menu=file_menu)

        # accelerator オプションでショートカット表示
        file_menu.add_command(label="新規", command=self.new_file, accelerator="Ctrl+N")
        file_menu.add_command(label="開く", command=self.open_file, accelerator="Ctrl+O")
        file_menu.add_command(label="保存", command=self.save_file, accelerator="Ctrl+S")
        file_menu.add_separator()
        file_menu.add_command(label="終了", command=root.quit, accelerator="Ctrl+Q")

        # 実際のキーバインディング（accelerator は表示のみ）
        root.bind('<Control-n>', lambda e: self.new_file())
        root.bind('<Control-o>', lambda e: self.open_file())
        root.bind('<Control-s>', lambda e: self.save_file())
        root.bind('<Control-q>', lambda e: root.quit())

        # underline オプションでアクセスキー
        edit_menu = tk.Menu(menubar, tearoff=0)
        menubar.add_cascade(label="編集", menu=edit_menu, underline=0)  # Eキー
        edit_menu.add_command(label="コピー", command=self.copy, underline=0)  # Cキー
        edit_menu.add_command(label="貼り付け", command=self.paste, underline=0)  # 貼キー

    def new_file(self):
        print("新規ファイル")

    def open_file(self):
        print("ファイルを開く")

    def save_file(self):
        print("ファイルを保存")

    def copy(self):
        print("コピー")

    def paste(self):
        print("貼り付け")
```

実践的な応用例

キーボードショートカット一覧表示

```
 import tkinter as tk
from tkinter import ttk

class ShortcutDisplayApp:
    def __init__(self, root):
        self.root = root
        root.title("キーボードショートカット例")

        # ショートカット一覧
        self.shortcuts = {
            'F1': 'ヘルプを表示',
            'Ctrl+N': '新規作成',
            'Ctrl+O': 'ファイルを開く',
            'Ctrl+S': '保存',
            'Ctrl+Z': '元に戻す',
            'Ctrl+Y': 'やり直し',
            'Esc': 'キャンセル',
            'Enter': '実行'
        }

        # UI作成
        self.create_ui()
        self.setup_bindings()

    def create_ui(self):
        # メインフレーム
        main_frame = tk.Frame(self.root)
        main_frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

        # ショートカット一覧表示
        tree = ttk.Treeview(main_frame, columns=('action',), show='tree headings')
        tree.heading('#0', text='ショートカット')
        tree.heading('action', text='動作')

        for shortcut, action in self.shortcuts.items():
            tree.insert('', tk.END, text=shortcut, values=(action,))
```

```
        tree.pack(fill=tk.BOTH, expand=True)

        # ステータスラベル
        self.status_label = tk.Label(main_frame, text="キーを押してください...",
                                     relief=tk.SUNKEN, anchor=tk.W)
        self.status_label.pack(fill=tk.X, pady=(10, 0))

    def setup_bindings(self):
        self.root.bind('<F1>', lambda e: self.show_action('ヘルプを表示'))
        self.root.bind('<Control-n>', lambda e: self.show_action('新規作成'))
        self.root.bind('<Control-o>', lambda e: self.show_action('ファイルを開く'))
        self.root.bind('<Control-s>', lambda e: self.show_action('保存'))
        self.root.bind('<Control-z>', lambda e: self.show_action('元に戻す'))
        self.root.bind('<Control-y>', lambda e: self.show_action('やり直し'))
        self.root.bind('<Escape>', lambda e: self.show_action('キャンセル'))
        self.root.bind('<Return>', lambda e: self.show_action('実行'))

    def show_action(self, action):
        self.status_label.config(text=f"実行: {action}")
        self.root.after(2000, lambda: self.status_label.config(text="キーを押してください..."))
```

まとめ

機能	説明	使用方法
基本バインディング	キーイベントを関数にバインド	`widget.bind('<Key>', callback)`
フォーカス移動	特定のウィジェットにフォーカスを設定	`widget.focus_set()`
グローバルバインディング	アプリケーション全体で有効なショートカット	`root.bind('<Control-s>', callback)`
コンテキストバインディング	特定のウィジェットでのみ有効	`text_widget.bind('<Control-d>', callback)`
アクセラレータ	メニューでのショートカット表示	`accelerator="Ctrl+S"`
アクセスキー	Altキーでのメニューアクセス	`underline=0`

ベストプラクティス

1. **一貫性**: 標準的なショートカット（Ctrl+C, Ctrl+V等）を使用
2. **表示**: `accelerator` オプションでユーザーにショートカットを示す
3. **コンテキスト**: ウィジェット固有のショートカットは適切なスコープで定義
4. **無効化**: 必要に応じてバインディングの有効/無効を切り替え
5. **ドキュメント**: 利用可能なショートカットをユーザーに明示