Qt Quick でのタブページの実装に手こずった話

妹尾 賢 (SENOO, Ken)

✉contact@senooken.jp

https://social.senooken.jp/senooken

2018-08-18

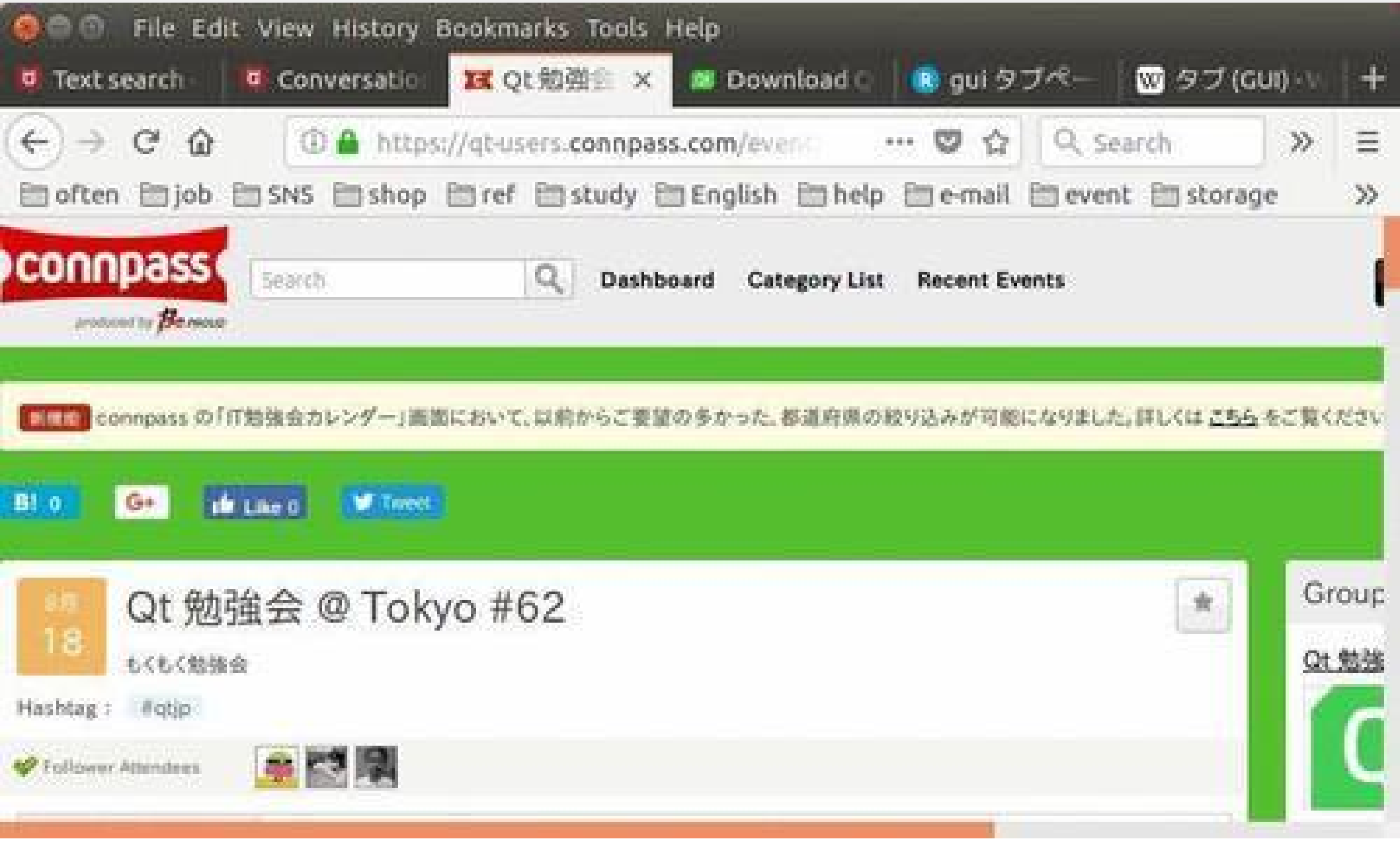
Qt 勉強会 @ Tokyo #62

<https://qt-users.connpass.com/event/97166/>

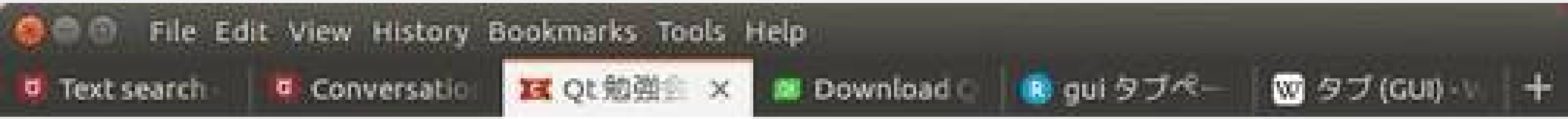
URL: https://senooken.jp/public/20180818/

Table of Contents

1. タブページとは


File Edit View History Bookmarks Tools Help
Text search
Conversatio
Qt勉強会
Download
gui タブペー
タブ (GUI)
https://qt-users.connpass.com/even
Search
often
job
SNS
shop
ref
study
English
help
e-mail
event
storage
connpass
produced by
Search
Dashboard
Category List
Recent Events
connpass の「IT勉強会カレンダー」画面において、以前からご要望の多かった、都道府県の絞り込みが可能になりました。詳しくは こちら をご覧ください
B! 0
G+
Like 0
Tweet
Qt 勉強会 @ Tokyo #62
もくもく勉強会
Hashtag :
#qtjp
Follower Attendees
Group
Qt 勉強

1. タブページとは



タブ（ tab ）とはドキュメントを切り替えて表示するための GUI ウィジェットである。一般的には長方形のボックス中にテキストラベルを表示する形で画面上部に表示され、タブの選択により管理するドキュメントを切り替えて表示させる仕組みとなっている。

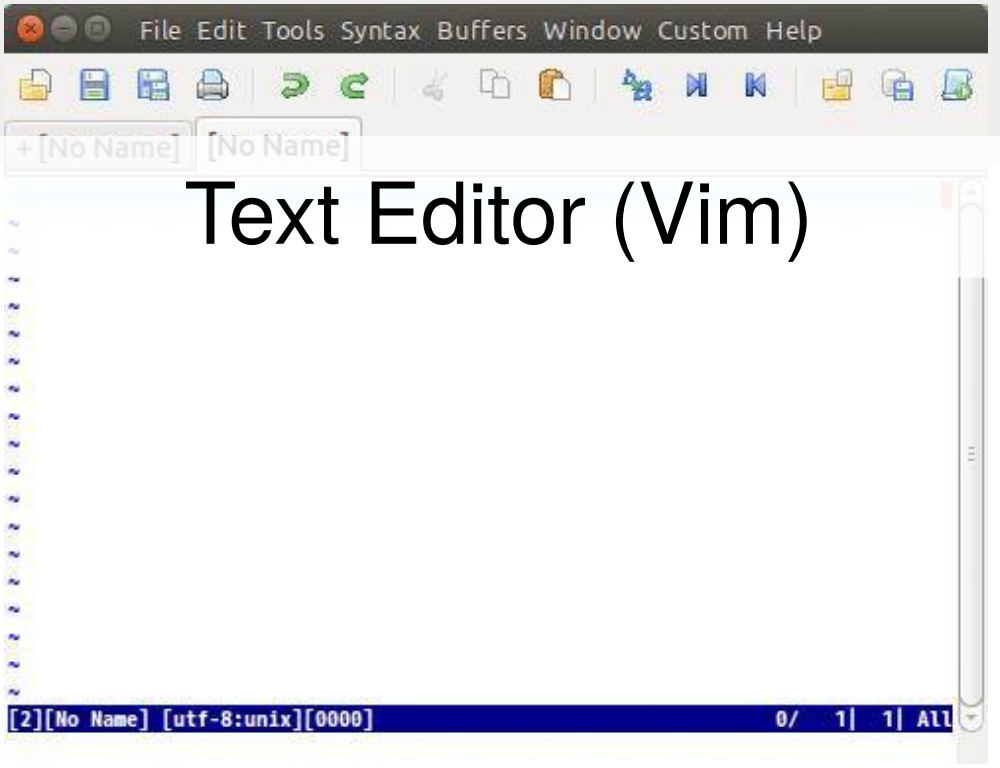
--Wikipedia | タブ (GUI)
< https://ja.wikipedia.org/wiki/タブ_(GUI)>

- 水平タブ (U+0009) と区別するため，**タブページ**とよぶ
- タブの見出し部分を**タブボタン**とよぶ

1. タブページとは

Web Browser (Firefox)

Text Editor (Vim)

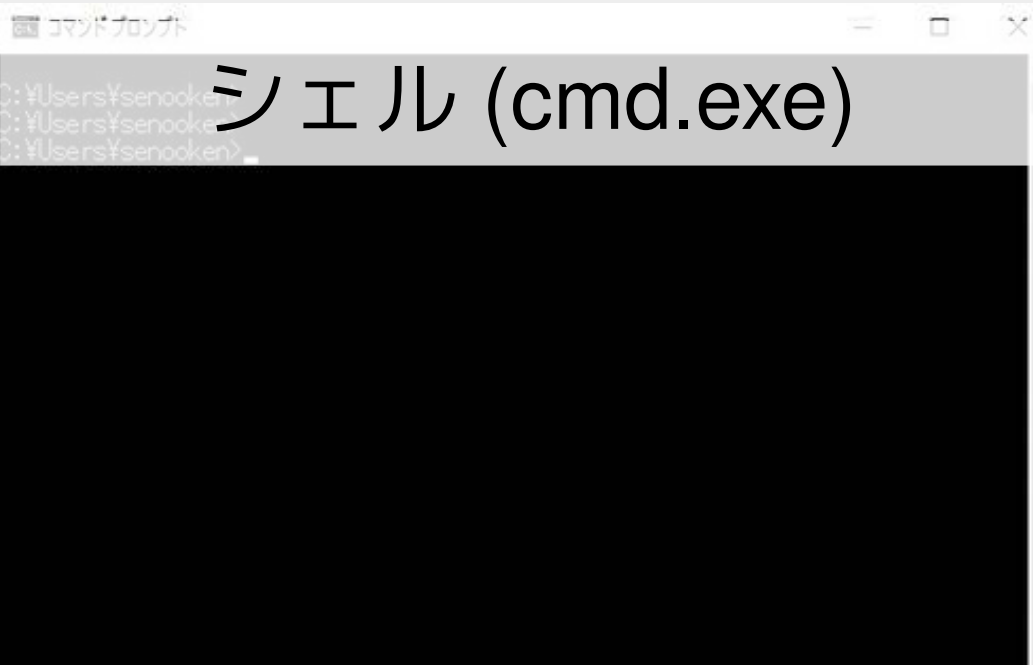


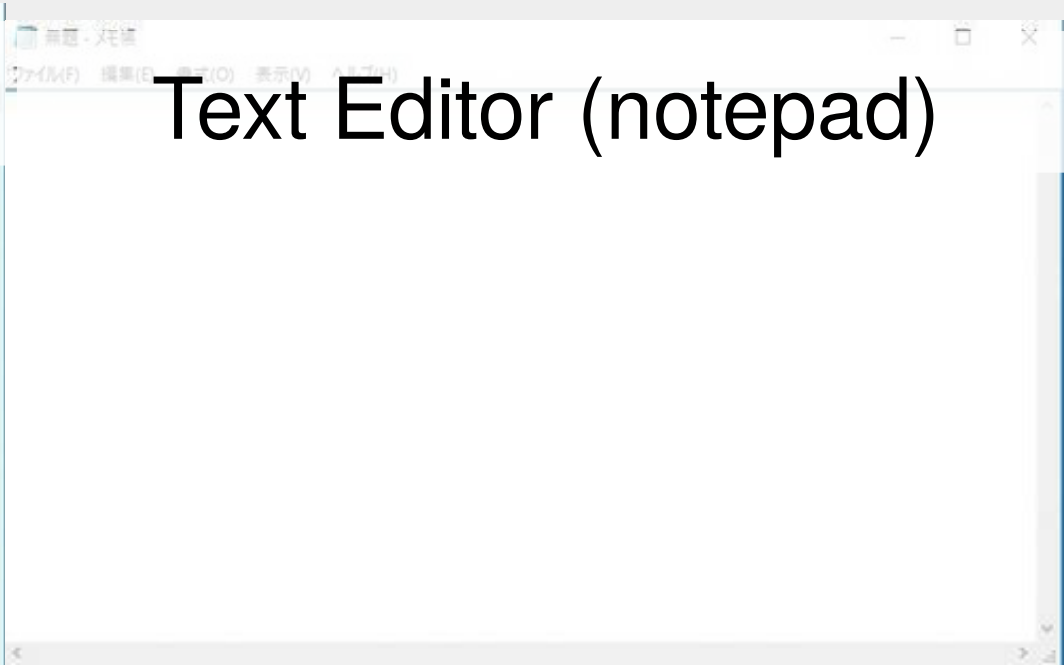
File Manager (Nautilus)

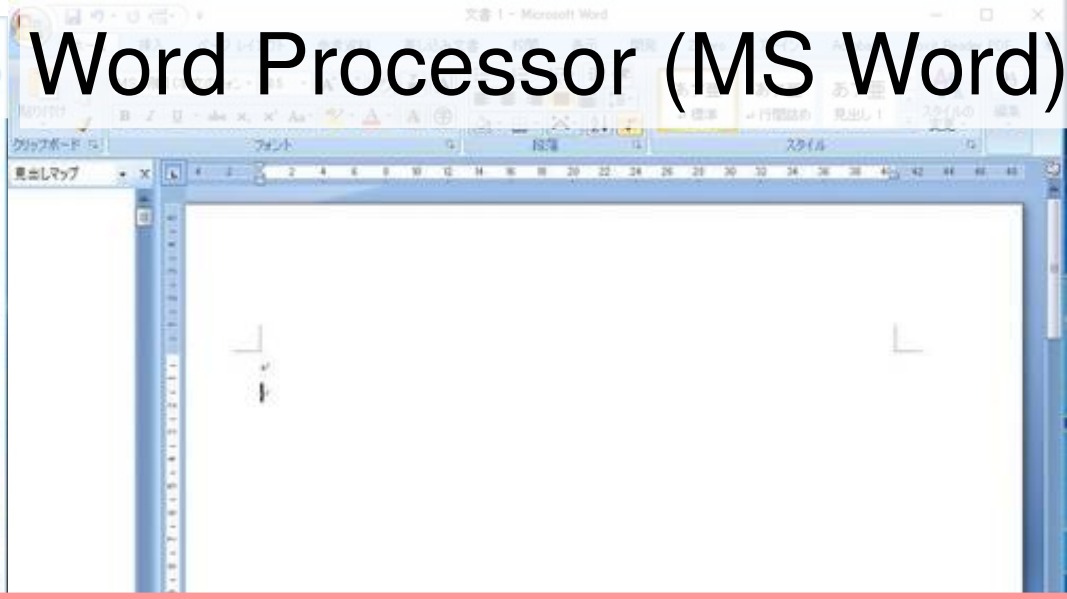
PDF Viewer (Foxit)

多数のソフトで採用されている GUI 部品

1. タブページとは

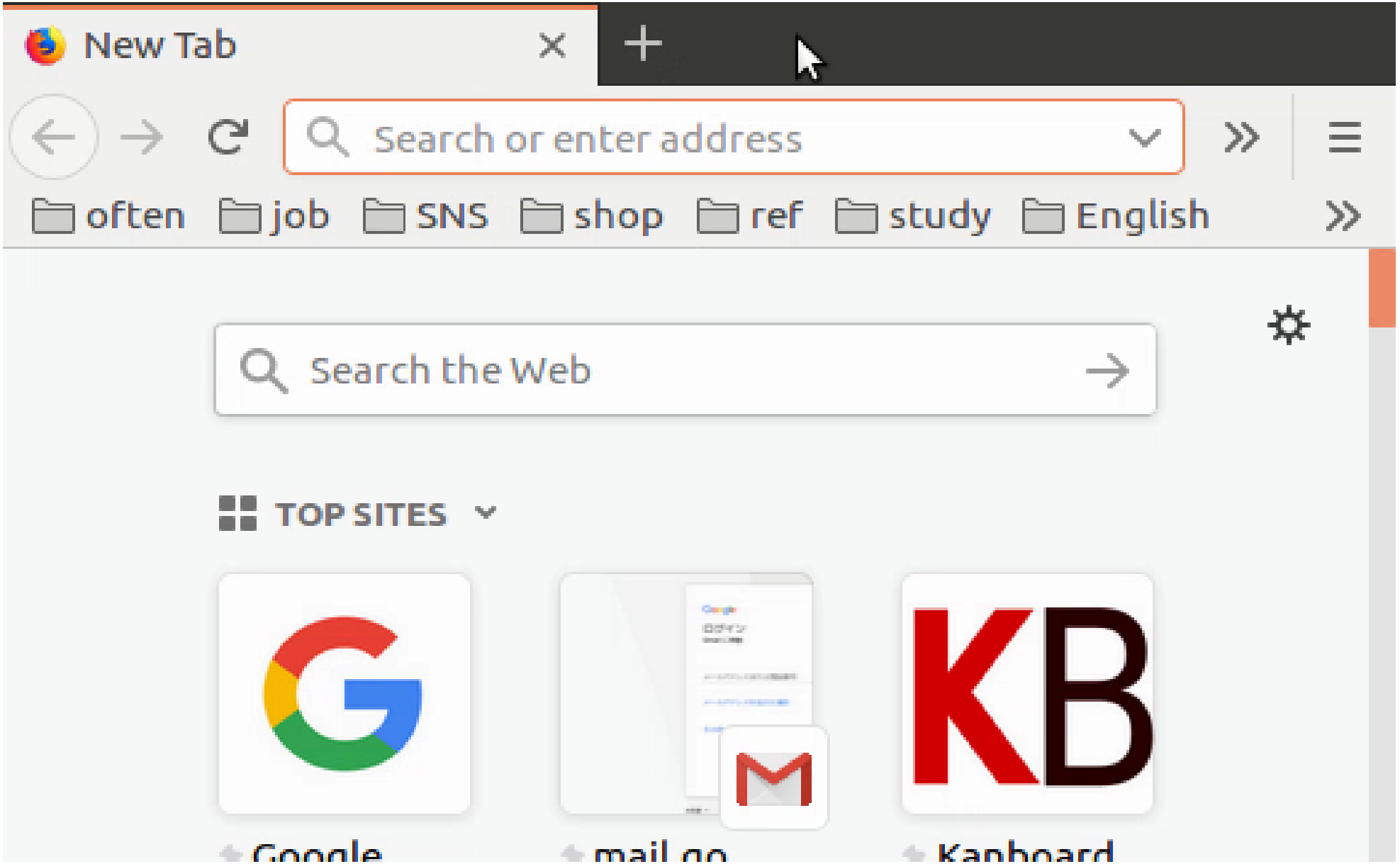






逆にタブページがないと**不便**

1. タブページの機能イメージ



1. タブページの機能

1.ボタン ([+]) でタブページ追加

2.マウスホバーで [×] ボタン表示

3.[×] ボタン押下でタブページ削除

4.タブボタンをドラッグで移動

5.タブボタンをドロップで別ウィンドウへ移動

GUI に必須のタブページを **Qt** で実装

2. Qt での 4 通りの開発方法

2. Qt での 4 通りの開発方法

1. Qt Widgets
2. Qt Quick
3. Qt Quick Controls 1
4. Qt Quick Controls 2

C++

QML

2. Qt での 4 通りの開発方法

1. Qt Widgets
2. ~~Qt Quick~~
3. ~~Qt Quick Controls 1~~
4. Qt Quick Controls 2〇

Qt Widgets or **Qt Quick Controls 2** の 2 択

今回は Qt Quick Controls 2 を選択

2. Qt での 4 通りの開発方法

1. Qt Widgets

   C++ で作る。昔から存在する Qt のベース。

2. Qt Quick

   Qt 5.0 から存在。 QML で作る。矩形描画など原始的機能提供。

   必要な部品は自作（非効率）。

3. Qt Quick Controls 1

   Qt 5.1 から存在。ダイアログなど GUI に必要な機能を提供。

   モバイル環境での性能が悪いため， **Qt 5.11 から廃止予定扱い**。

   Qt Wiki: https://wiki.qt.io/New_Features_in_Qt_5.11

   ML: http://lists.qt-project.org/pipermail/development/2018-February/032073.html

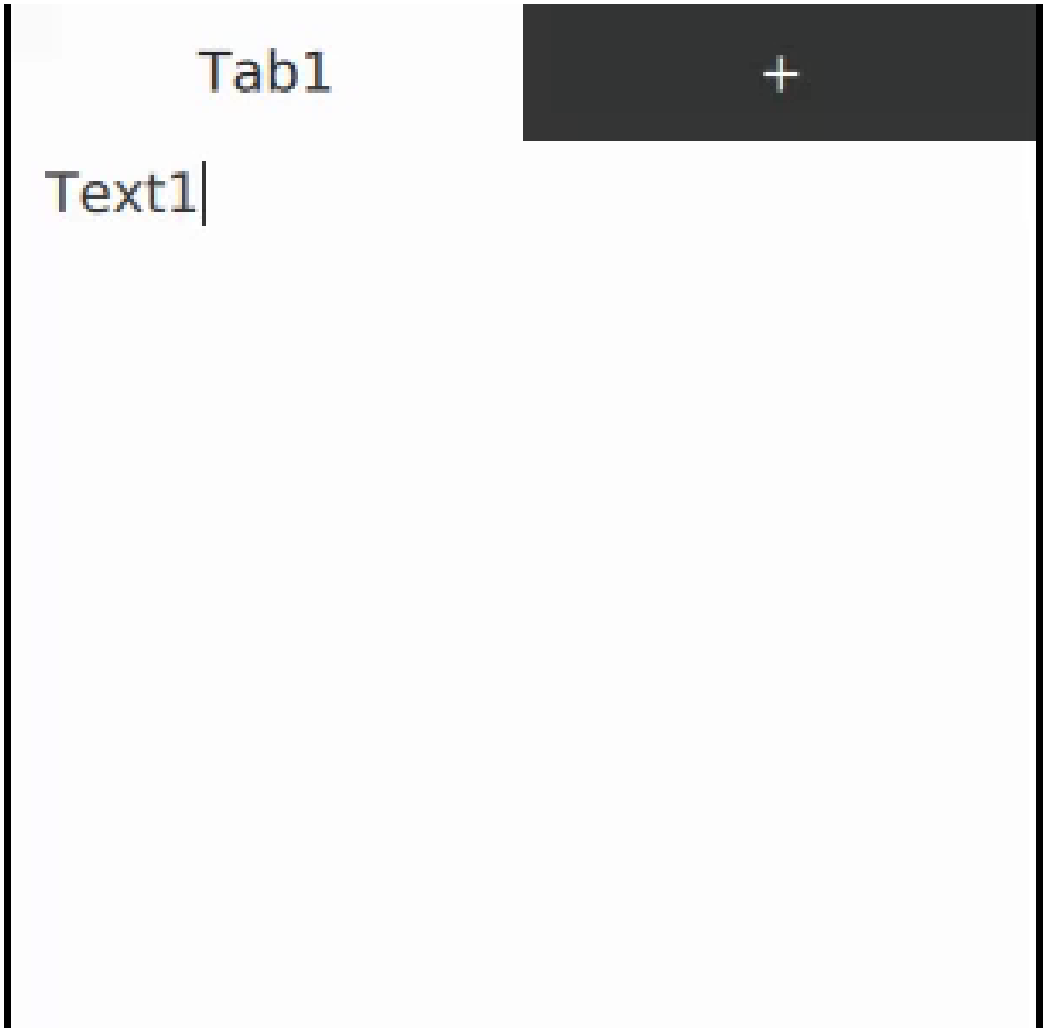
4. Qt Quick Controls 2

   Qt 5.7 から存在。 1 系での性能を改善。

   モバイルフレンドリー？

3. Qt Quick Controls 2 での開発

3. Qt Quick Controls 2 での実装動作イメージ



3. Qt Quick Controls 2 での開発

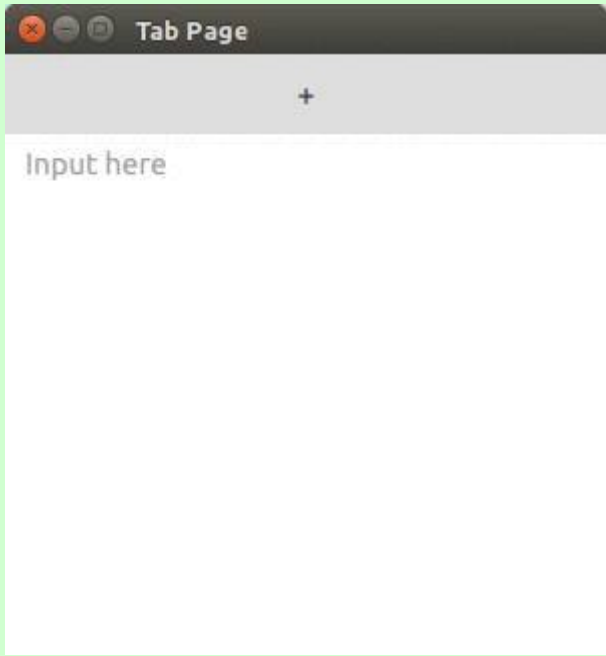
- Qt 5.10.0 で開発
- 開発期間 : 2018-07-14~08-15
- ソースコード :
  https://github.com/senooken/QtExample/tree/master/TabPageQML
- main.qml の 1 ファイルでのシンプル実装
- ドロップで別画面への移動は未実装
- タブボタンのダブルクリックでリネームを実装
- こちらを主に参考にした
  Adding TabButton dynamically to TabBar | Qt Forum
  https://forum.qt.io/topic/81768/adding-tabbutton-dynamically-to-tabbar/10

1. タブページ外観と追加 ([+]) ボタン

タブページの実装のための TabBar と TabButton QML Type を利用

- https://doc.qt.io/qt-5/qml-qtquick-controls2-tabbar.html
- https://doc.qt.io/qt-5/qml-qtquick-controls2-tabbutton.html

```
import QtQuick 2.10
import QtQuick.Controls 2.0

ApplicationWindow {
    id: window
    visible: true
    title: "Tab Page"
    width: 300
    height: 300

    header: TabBar {
        id: tabBar
        currentIndex: view.currentIndex
        TabButton {
            id: addButton
            text: "+"
            onClicked: addTab()
        }
    }

    SwipeView {
        id: view
        anchors.fill: parent
        currentIndex: tabBar.currentIndex
        interactive: false
        TextArea {placeholderText: "Input here"}
    }
}
```



ヘッダーとコンテントを連動

[+] ボタン部

後でイベントハンドラー定義

- サンプルでは Repeater を多用。しかし，タブページ削除機能の都合断念。
- 同種の StackView と ScrollView は Control 型。 View で唯一 Container 型の SwipeView が都合いい (後述) 。

2. タブページ追加の実装

QML でのオブジェクトの動的生成・削除の一般的方法

- Dynamic QML Object Creation from JavaScript
- http://doc.qt.io/qt-5/qtqml-javascript-dynamicobjectcreation.html

Qt Quick Controls 2 の Container 型 (TabBar と SwipeView) は動的生成・削除のメソッド (addItem, insertItem, removeItem) があり簡単

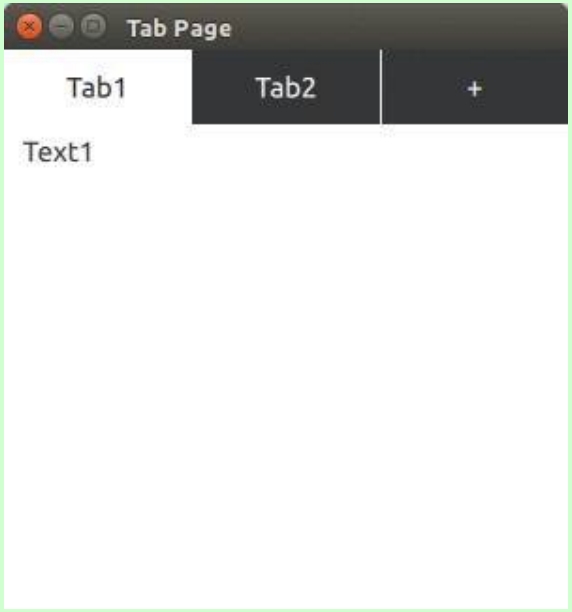
- https://doc.qt.io/qt-5/qml-qtquick-controls2-container.html

```
ApplicationWindow {
    function addTab() {
        tabBar.insertItem(tabBar.currentIndex, tabButton.createObject(tabBar, {text: "Tab"+(tabBar.currentIndex+1)}))
        tabBar.setCurrentIndex(tabBar.currentIndex-1)
        view.insertItem(tabBar.currentIndex, tabContent.createObject(view, {text: "Text"+ (tabBar.currentIndex+1)}))
        view.setCurrentIndex(view.currentIndex-1)
    }

    Component.onCompleted: addTab()

// header: TabBar {...}
// SwipeView {...}

    Component {
        id: tabContent
        TextArea {placeholderText: "Input here"}
    }

    Component {
        id: tabButton
        TabButton { /* ここが長い */ }
    }
}
```

[+] の直前にタブボタンとコンテンツを挿入

先頭の Tab1 を最初に生成

動的生成用コンテンツを用意



3. 閉じるボタン ([×]) の実装

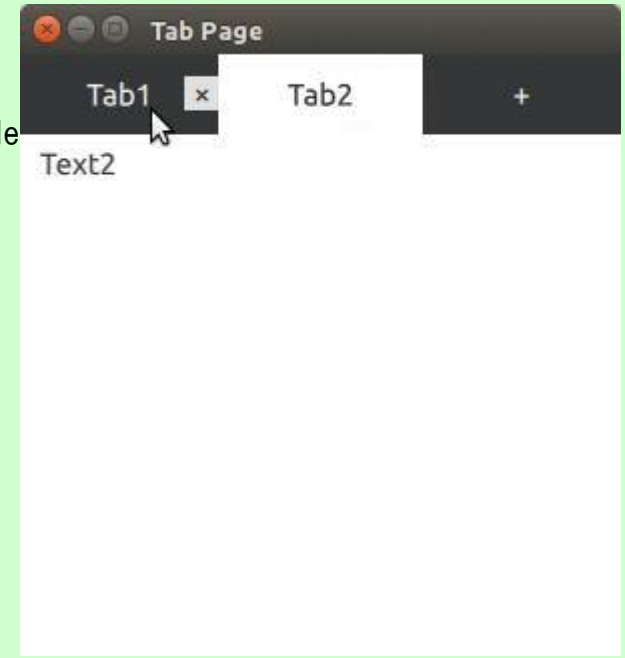
- マウスホバーで閉じるボタン ([×]) を表示 （非標準機能のため自作）
- MouseArea 内にホバーしたときだけ ×Button を表示

```
Component {
    id: tabButton
    TabButton {
        // Avoid moving right focus to add tab button.
        Keys.onRightPressed: {
            if ( tabBar.currentIndex+2 < tabBar.count ) tabBar.incrementCurrentInde
        }

        MouseArea {
            id: mouseArea
            anchors.fill: parent
            visible: true
            hoverEnabled: true

            onEntered: closeButton.visible = true
            onExited: closeButton.visible = false

            Button {
                id: closeButton
                anchors.right: parent.right
                anchors.verticalCenter: parent.verticalCenter
                visible: false
                FontMetrics {id: fm}
                width: fm.height * closeButton.text.length
                height: fm.height
                text: "×"
                onClicked: {parent.closeTab(parent)}
            }
        }

    }
}
```

ホバー中だけ [×] を表示

[×] のフォントサイズ・配置調整

後でイベントハンドラー定義



4. 削除の実装

```
MouseArea {
      ...
      hoverEnabled: true

      property int nowIndex: 0
      property int hoveredIndex : 0
      property int newIndex : 0

      function closeTab(parent) {
            view.removeItem(view.itemAt(mouseArea.hoveredIndex))
            tabBar.removeItem(tabBar.itemAt(mouseArea.hoveredIndex))
            if (hoveredIndex == 0) tabBar.setCurrentIndex(0)
            // updateTabX() //  タブページ削除時の座標更新
      }

      function updateTabPosition() {
            newIndex = tabBar.currentIndex

            if ((mouseX < 0) && (tabBar.currentIndex > 0)) {
                  newIndex = tabBar.currentIndex - 1;
            } else if ((mouseX > width-1) && (tabBar.currentIndex+2 < tabBar.count)) {
                  newIndex = tabBar.currentIndex + 1;
            }

            // Save current hovered tab index
            var windowPosition = mapToItem(tabBar, mouseX, mouseY)
            for (var i in tabBar.contentChildren) {
                  var tab = tabBar.contentChildren[i]
                  if ((tab.x <= windowPosition.x) && (windowPosition.x <= (tab.x+tab.width))) {
                        hoveredIndex = i;
                  }
            }
      }

      onPressed: {
            tabBar.setCurrentIndex(hoveredIndex)
            nowIndex = hoveredIndex
      }

      // Show close button
      onEntered: {
            updateTabPosition()
            closeButton.visible = true
      }

      onExited: closeButton.visible = false
      Button { ... }
}
```

マウスホバー時のタブボタンの取得

- ここから急に難易度上昇
- 現在のタブページ以外でも[×] 押下時に削除するため，マウスホバー時のタブボタンの（添字）取得が必要
- マウス座標とタブボタンの座標を比較してマウスがタブボタン上か判定

5. タブボタンのドラッグ

```
function updateTabPosition() {
	...
	// Save current hovered tab index
	var windowPosition = mapToItem(tabBar, mouseX, mouseY)
	for (var i in tabBar.contentChildren) {
		var tab = tabBar.contentChildren[i]
		if ((tab.x <= windowPosition.x) && (windowPosition.x <=
(tab.x+tab.width))) {
			hoveredIndex = i;
		}
	}

	if (drag.active) {
		// Tab position switching condition
		if ((nowIndex > 0) && (tabBar.contentChildren[nowIndex].x
<= tabBar.contentChildren[nowIndex-1].x)) {
			newIndex = nowIndex - 1
		} else if ((nowIndex < tabBar.count-2) &&
(tabBar.contentChildren[nowIndex].x >=
tabBar.contentChildren[nowIndex+1].x)) {
			newIndex = nowIndex + 1
		}

		// Update tab position
		if (nowIndex != newIndex) {
			tabBar.moveItem(nowIndex, newIndex)
			view.moveItem(nowIndex, newIndex)
			tabBar.setCurrentIndex(newIndex)
			view.setCurrentIndex(newIndex)
			nowIndex = newIndex
		}
	}
}

drag.target: parent
drag.axis: Drag.XAxis

// Drag and move tab
onPositionChanged: {
	updateTabPosition()
}

// When released drag, fixing tab position.
onReleased: {
	if (!drag.active) return

	var rightItem = tabBar.itemAt(currentIndex+1)
	tabBar.currentItem.x = rightItem.x - tabBar.currentItem.width
	// updateTabX() //  タブ位置更新時座標異常対応
}
...
// Show close button
onEntered: {...}
...
```

- タブドラッグ中に隣のタブを超えたら移動
- ドラッグ解放時に位置を固定

x 方向ドラッグ有効化

ドラッグ中のタブの交換

ドラッグ解放時位置固定
本来なら直前の 2 行 (右隣のタブの隣に固定) で済むはずだが，クリックの位置がおかしくなるので updateX() を使う

6. タブタイトルの編集機能

- タブボタンをダブルクリックでタイトルを編集可能にする
- TabButton に TextField を重ねてダブルクリック時だけ有効化

```
TabButton {
    // Avoid moving right focus to add tab button.
    Keys.onRightPressed: {
        if ( tabBar.currentIndex+2 < tabBar.count ) tabBar.incrementCurrentIndex()
    }
    // Rename tab page title
    TextField {
        id: textField
        anchors.fill: parent
        horizontalAlignment: TextInput.AlignHCenter
        visible: false
        text: parent.text

        onEditingFinished: {
            parent.text = text
            textField.visible = false
            mouseArea.visible = true
        }
    }
    ...
    MouseArea {
        ...
        onPressed: {
            tabBar.setCurrentIndex(hoveredIndex)
            nowIndex = hoveredIndex
        }

        // Change focus
        onDoubleClicked: {
            tabBar.setCurrentIndex(hoveredIndex)
            mouseArea.visible = false
            textField.visible = true
            textField.focus = true
        }

        // Show close button
        onEntered: { ... }
```

タイトル編集後 TextField 非表示
MouseArea 表示

ダブルクリック時 MouseArea 非表示 TextField 表示

4. 問題点

4.1. タブページの削除・移動時に座標がおかしくなる

以下の操作後，タブボタンをクリックしたときの位置がずれる。

- [×] で削除時
- タブボタンをドラッグで移動時

> 対策として updateX() で無理やり TabButton の座標を固定。

また，タブを高速でドラッグして移動させるとタブの配置がおかしくなる。

> 対策として幅を更新

```
MouseArea {
...
// After drag, update position is wrong
function updateTabX() {
     for (var i = 0, len = tabBar.contentChildren.length, width = tabBar.width/len; i <
len; ++i) {
          tabBar.contentChildren[i].x =  i * width
     }
     // Updating window size for alignment tab button.
     window.width += 1
     var start = new Date().getTime()
     var stop = new Date().getTime()
     while (stop < start + 20) {
          stop = new Date().getTime()
     }
     window.width -= 1
}
```

幅更新。早いと反映されないので 20 ms 待機。

4.2. 起動直後のタブボタン編集時にフォーカズあわない

- 起動直後，最初のタブボタンをダブルクリックしても，フォーカスされず TextField の編集に入らない (カーソルが表示されない) 。
- もう一度クリックすると編集できる
- タブページを追加した場合などは問題ない。

4.3. Qt 5.11 でタブボタン編集時に文字が被る

- タブボタンをダブルクリックすると，一番上の TextField が表示されて， TabButton は隠れるはず。
- Qt 5.11 だとなぜか下の TabButton のテキストが隠れずに表示され，座標が微妙にずれて二重に表示される

まとめ

- Qt Quick Controls 2 でのタブページの実装を検討
- 複雑なこと (以下 2 点) をしない場合，簡単に実装可能
  - タブの移動を不許可
  - タブの削除は現在タブのみ
- 標準外の実装は困難
  - 動的生成・削除，移動などはバグらしき挙動
  - Controls 2 は新しい (Qt 5.7 から) ので不安定？
  - 割り切って，設計で複雑な実装をしないようにする？
- 今後， Qt Widgets で同じ機能を実装し， Qt Quick Controls 2 と比較検討したい

Q & A

1. Qt Quick Controls 2 でのタブページの実装例がほぼない

   Qt Quick Controls 1 だとネット上に情報があった気がする。ただし，ドラッグはできなかった。

2. Controls 2 はまだ安定していないのか？

   だいぶ安定してきているとは思うが，不具合なのか，なんだろうね？

3. ドラッグさせない，現在タブだけ閉じるとか機能を限定したら，たしかに Controls 2 はスマートにできると感じた。しかし，ちょっと凝ったことをすると急に難しい

   提供されていない機能を実装するのはたしかに難しい

Q & A

4.今回の実装は Android などで動くか試したか？

いいえ。不具合があるので，デスクトップ環境でちゃんと動作するものを作ってから， Android などでの動作を検証する。

今後， Qt Quick と Qt Widgets のどちらメインに使うか検討中なので，それらが落ち着いてからになる。