atom / **atom**

Branch: master ▾ atom / src / **menu-manager.coffee** Find file Copy path

rafeca Merge branch 'master' into electron-4 5ddb665 on 15 Jun 2019

10 contributors

212 lines (182 sloc) 6.5 KB Raw Blame History

```
path = require 'path'

_ = require 'underscore-plus'
{ipcRenderer} = require 'electron'
CSON = require 'season'
fs = require 'fs-plus'
{Disposable} = require 'event-kit'

MenuHelpers = require './menu-helpers'

platformMenu = require('../package.json')?._atomMenu?.menu

# Extended: Provides a registry for menu items that you'd like to appear in the
# application menu.
#
# An instance of this class is always available as the `atom.menu` global.
#
# ## Menu CSON Format
#
# Here is an example from the [tree-view](https://github.com/atom/tree-view/blob/mas
#
```

```coffee
# ```coffee
# [
#   {
#     'label': 'View'
#     'submenu': [
#       { 'label': 'Toggle Tree View', 'command': 'tree-view:toggle' }
#     ]
#   }
#   {
#     'label': 'Packages'
#     'submenu': [
#       'label': 'Tree View'
#       'submenu': [
#         { 'label': 'Focus', 'command': 'tree-view:toggle-focus' }
#         { 'label': 'Toggle', 'command': 'tree-view:toggle' }
#         { 'label': 'Reveal Active File', 'command': 'tree-view:reveal-active-file'
#         { 'label': 'Toggle Tree Side', 'command': 'tree-view:toggle-side' }
#       ]
#     ]
#   }
# ]
# ```
#
# Use in your package's menu `.cson` file requires that you place your menu
# structure under a `menu` key.
#
# ```coffee
# 'menu': [
#   {
#     'label': 'View'
#     'submenu': [
#       { 'label': 'Toggle Tree View', 'command': 'tree-view:toggle' }
#     ]
#   }
# ]
# ```
#
# See {::add} for more info about adding menu's directly.
module.exports =
class MenuManager
  constructor: ({@resourcePath, @keymapManager, @packageManager}) ->
    @initialized = false
    @pendingUpdateOperation = null
    @template = []
    @keymapManager.onDidLoadBundledKeymaps => @loadPlatformItems()
```

```coffee
    @packageManager.onDidActivateInitialPackages => @sortPackagesMenu()

  initialize: ({@resourcePath}) ->
    @keymapManager.onDidReloadKeymap => @update()
    @update()
    @initialized = true

  # Public: Adds the given items to the application menu.
  #
  # ## Examples
  # ```coffee
  #   atom.menu.add [
  #     {
  #       label: 'Hello'
  #       submenu : [{label: 'World!', command: 'hello:world'}]
  #     }
  #   ]
  # ```
  #
  # * `items` An {Array} of menu item {Object}s containing the keys:
  #   * `label` The {String} menu label.
  #   * `submenu` An optional {Array} of sub menu items.
  #   * `command` An optional {String} command to trigger when the item is
  #     clicked.
  #
  # Returns a {Disposable} on which `.dispose()` can be called to remove the
  # added menu items.
  add: (items) ->
    items = _.deepClone(items)

    for item in items
      continue unless item.label? # TODO: Should we emit a warning here?
      @merge(@template, item)

    @update()
    new Disposable => @remove(items)

  remove: (items) ->
    @unmerge(@template, item) for item in items
    @update()

  clear: ->
    @template = []
    @update()

```

```
  # Should the binding for the given selector be included in the menu
  # commands.
  #
  # * `selector` A {String} selector to check.
  #
  # Returns a {Boolean}, true to include the selector, false otherwise.
  includeSelector: (selector) ->
    try
      return true if document.body.webkitMatchesSelector(selector)
    catch error
      # Selector isn't valid
      return false

    # Simulate an atom-text-editor element attached to a atom-workspace element atta
    # to a body element that has the same classes as the current body element.
    unless @testEditor?
      # Use new document so that custom elements don't actually get created
      testDocument = document.implementation.createDocument(document.namespaceURI, '

      testBody = testDocument.createElement('body')
      testBody.classList.add(@classesForElement(document.body)...)

      testWorkspace = testDocument.createElement('atom-workspace')
      workspaceClasses = @classesForElement(document.body.querySelector('atom-worksp
      workspaceClasses = ['workspace'] if workspaceClasses.length is 0
      testWorkspace.classList.add(workspaceClasses...)

      testBody.appendChild(testWorkspace)

      @testEditor = testDocument.createElement('atom-text-editor')
      @testEditor.classList.add('editor')
      testWorkspace.appendChild(@testEditor)

    element = @testEditor
    while element
      return true if element.webkitMatchesSelector(selector)
      element = element.parentElement

    false

  # Public: Refreshes the currently visible menu.
  update: ->
    return unless @initialized

    clearTimeout(@pendingUpdateOperation) if @pendingUpdateOperation?

    @pendingUpdateOperation = setTimeout(=>
      unsetKeystrokes = new Set
      for binding in @keymapManager.getKeyBindings()
        if binding.command is 'unset!'
          unsetKeystrokes.add(binding.keystrokes)

      keystrokesByCommand = {}
      for binding in @keymapManager.getKeyBindings()
        continue unless @includeSelector(binding.selector)
        continue if unsetKeystrokes.has(binding.keystrokes)
        continue if process.platform is 'darwin' and /^alt-(shift-)?.$/.test(binding
        continue if process.platform is 'win32' and /^ctrl-alt-(shift-)?.$/.test(bin
        keystrokesByCommand[binding.command] ?= []
        keystrokesByCommand[binding.command].unshift binding.keystrokes

      @sendToBrowserProcess(@template, keystrokesByCommand)
    , 1)

  loadPlatformItems: ->
    if platformMenu?
      @add(platformMenu)
    else
      menusDirPath = path.join(@resourcePath, 'menus')
      platformMenuPath = fs.resolve(menusDirPath, process.platform, ['cson', 'json']
      {menu} = CSON.readFileSync(platformMenuPath)
      @add(menu)

  # Merges an item in a submenu aware way such that new items are always
  # appended to the bottom of existing menus where possible.
  merge: (menu, item) ->
    MenuHelpers.merge(menu, item)

  unmerge: (menu, item) ->
    MenuHelpers.unmerge(menu, item)

  sendToBrowserProcess: (template, keystrokesByCommand) ->
    ipcRenderer.send 'update-application-menu', template, keystrokesByCommand

  # Get an {Array} of {String} classes for the given element.
  classesForElement: (element) ->
    if classList = element?.classList
      Array::slice.apply(classList)
    else
      []

  sortPackagesMenu: ->
    packagesMenu = _.find @template, ({label}) -> MenuHelpers.normalizeLabel(label)
    return unless packagesMenu?.submenu?

    packagesMenu.submenu.sort (item1, item2) ->
      if item1.label and item2.label
        MenuHelpers.normalizeLabel(item1.label).localeCompare(MenuHelpers.normalizeL
      else
        0
    @update()
```