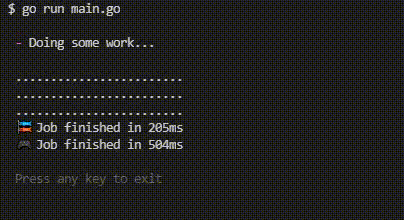
简介

[bubbletea](https://github.com/charmbracelet/bubbletea)是一个简单、小巧、可以非常方便地用来编写 TUI（terminal User Interface，控制台界面程序）程序的框架。内置简单的事件处理机制，可以对外部事件做出响应，如键盘按键。一起来看下吧。先看看bubbletea能做出什么效果：



感谢[kiyonlin](https://github.com/kiyonlin)推荐。

快速使用

本文代码使用 Go Modules。

创建目录并初始化：

$ mkdir bubbletea && **cd** bubbletea

$ **go** **mod** init github.**com**/darjun/**go**-daily-lib/bubbletea

安装bubbletea库：

$ **go** get -**u** github.**com**/charmbracelet/bubbletea

bubbletea程序都需要有一个实现bubbletea.Model接口的类型：

**type** **Model** interface {

**Init**() **Cmd**

**Update(Msg)** (**Model**, **Cmd**)

**View**() string

}

* Init()方法在程序启动时会立刻调用，它会做一些初始化工作，并返回一个Cmd告诉bubbletea要执行什么命令；
* Update()方法用来响应外部事件，返回一个修改后的模型，和想要bubbletea执行的命令；
* View()方法用于返回在控制台上显示的文本字符串。

下面我们来实现一个 Todo List。首先定义模型：

type model **struct** {

todos []string

cursor **int**

selected map[**int**]**struct**{}

}

* todos：所有待完成事项；
* cursor：界面上光标位置；
* selected：已完成标识。

不需要任何初始化工作，实现一个空的Init()方法，并返回nil：

**import** (

tea "github.com/charmbracelet/bubbletea"

)

**func** (m model) **Init**() **tea**.**Cmd** {

**return** nil

}

我们需要响应按键事件，实现Update()方法。按键事件发生时会以相应的tea.Msg为参数调用Update()方法。通过对参数tea.Msg进行类型断言，我们可以对不同的事件进行对应的处理：

**func** (m model) **Update**(msg tea.Msg) (tea.Model, tea.Cmd) {

**switch** msg := msg.(**type**) {

**case** tea.KeyMsg:

**switch** msg.String() {

**case** "ctrl+c", "q":

**return** m, tea.Quit

**case** "up", "k":

**if** m.cursor > 0 {

m.cursor--

}

**case** "down", "j":

**if** m.cursor < len(m.todos)-1 {

m.cursor++

}

**case** "enter", " ":

\_, ok := m.selected[m.cursor]

**if** ok {

delete(m.selected, m.cursor)

} **else** {

m.selected[m.cursor] = **struct**{}{}

}

}

}

**return** m, nil

}

约定：

* ctrl+c或q：退出程序；
* up或k：向上移动光标；
* down或j：向下移动光标；
* enter或 ：切换光标处事项的完成状态。

处理ctrl+c或q按键时，返回一个特殊的tea.Quit，通知bubbletea需要退出程序。

最后实现View()方法，这个方法返回的字符串就是最终显示在控制台上的文本。我们可以按照自己想要的形式，根据模型数据拼装：

**func** (m model) **View**() **string** {

s := "todo list:\n\n"

**for** i, choice := **range** m.todos {

cursor := " "

**if** m.cursor == i {

cursor = ">"

}

checked := " "

**if** \_, ok := m.selected[i]; ok {

checked = "x"

}

s += fmt.Sprintf("%s [%s] %s\n", cursor, checked, choice)

}

s += "\nPress q to quit.\n"

**return** s

}

光标所在位置用>标识，已完成的事项增加x标识。

模型类型定义好了之后，需要创建一个该模型的对象；

**var** initModel = model{

todos: []**string**{"cleanning", "wash clothes", "write a blog"},

selected: make(**map**[**int**]**struct**{}),

}

为了让程序工作，我们还要创建一个bubbletea的应用对象，通过bubbletea.NewProgram()完成，然后调用这个对象的Start()方法开始执行：

**func** **main**() {

cmd := tea.NewProgram(initModel)

**if** err := cmd.Start(); err != nil {

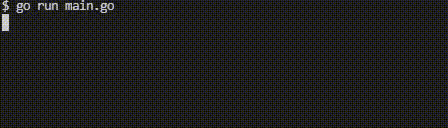
fmt.Println("start failed:", err)

os.Exit(1)

}

}

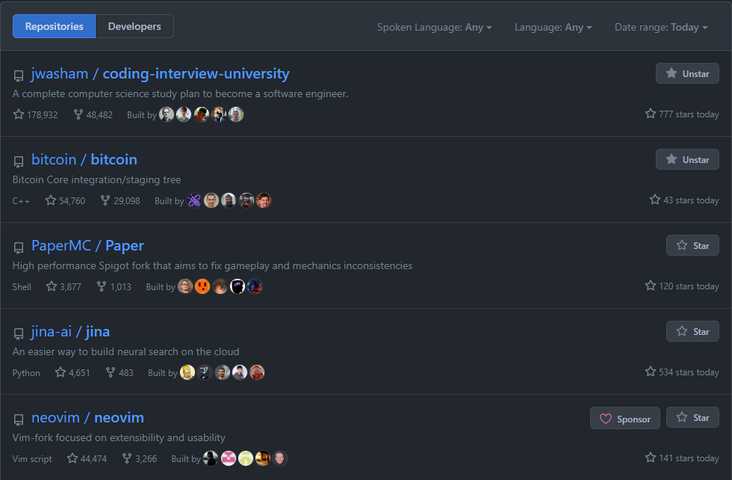
运行：



GitHub Trending

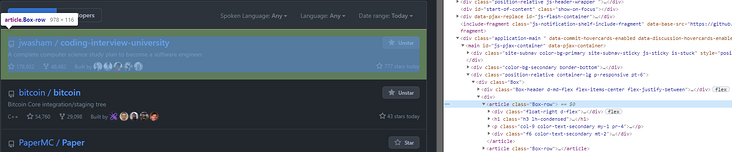
一个简单的 Todo 应用看起来好像没什么意思。接下来，我们一起编写一个拉取 GitHub Trending 仓库并显示在控制台的程序。

Github Trending 的界面如下：



可以选择语言（Spoken Language，本地语言）、语言（Language，编程语言）和时间范围（Today，This week，This month）。由于 GitHub 没有提供 trending 的官方 API，我们只能爬取网页自己来分析。好在 Go 有一个强大的分析工具[goquery](https://github.com/PuerkitoBio/goquery" \t "_blank)，提供了堪比 jQuery 的强大功能。我之前也写过一篇文章介绍它——[Go 每日一库之 goquery](https://darjun.github.io/2020/10/11/godailylib/goquery)。

打开 Chrome 控制台，点击 Elements 页签，查看每个条目的结构：



基础版本

定义模型：

**type** model struct {

repos []\***Repo**

err error

}

其中repos字段表示拉取到的 Trending 仓库列表，结构体Repo如下，字段含义都有注释，很清晰了：

**type** Repo **struct** {

Name **string** *// 仓库名*

Author **string** *// 作者名*

Link **string** *// 链接*

Desc **string** *// 描述*

Lang **string** *// 语言*

Stars **int** *// 星数*

Forks **int** *// fork 数*

Add **int** *// 周期内新增*

BuiltBy []**string** *// 贡献值 avatar img 链接*

}

err字段表示拉取失败设置的错误值。为了让程序启动时，就去执行网络请求拉取 Trending 的列表，我们让模型的Init()方法返回一个tea.Cmd类型的值：

**func** (m model) **Init**() **tea**.**Cmd** {

**return** fetchTrending

}

**func** **fetchTrending**() **tea**.**Msg** {

repos, err := getTrending("", "daily")

**if** err != nil {

**return** errMsg{err}

}

**return** repos

}

tea.Cmd类型为：

// src/github.com/charmbracelet/bubbletea/tea.go

type Cmd **func**() Msg

tea.Cmd底层是一个函数类型，函数无参数，并且返回一个tea.Msg对象。

fetchTrending()函数拉取 GitHub 的今日 Trending 列表，如果遇到错误，则返回error值。这里我们暂时忽略getTrending()函数的实现，这个与我们要说的重点关系不大，感兴趣的童鞋可以去我的 GitHub 仓库查看详细代码。

程序启动时如果需要做一些操作，通常就会在Init()方法中返回一个tea.Cmd。tea后台会执行这个函数，最终将返回的tea.Msg传给模型的Update()方法。

**func** (m model) **Update**(msg tea.Msg) (tea.Model, tea.Cmd) {

**switch** msg := msg.(**type**) {

**case** tea.KeyMsg:

**switch** msg.String() {

**case** "q", "ctrl+c", "esc":

**return** m, tea.Quit

**default**:

**return** m, nil

}

**case** errMsg:

m.err = msg

**return** m, nil

**case** []\*Repo:

m.repos = msg

**return** m, nil

**default**:

**return** m, nil

}

}

Update()方法也比较简单，首先还是需要监听按键事件，我们约定按下 q 或 ctrl+c 或 esc 退出程序。**具体按键对应的字符串表示可以查看文档或源码bubbletea/key.go文件**。接收到errMsg类型的消息，表示网络请求失败了，记录错误值。接收到[]\*Repo类型的消息，表示正确返回的 Trending 仓库列表，记录下来。在View()函数中，我们显示正在拉取，拉取失败和正确拉取等信息：

**func** (m model) **View**() **string** {

**var** s **string**

**if** m.err != nil {

s = fmt.Sprintf("Fetch trending failed: %v", m.err)

} **else** **if** len(m.repos) > 0 {

**for** \_, repo := **range** m.repos {

s += repoText(repo)

}

s += "--------------------------------------"

} **else** {

s = " Fetching GitHub trending ..."

}

s += "\n\n"

s += "Press q or ctrl + c or esc to exit..."

**return** s + "\n"

}

逻辑很清晰，如果err字段不为nil表示失败，否则有仓库数据，显示仓库信息。否则正在拉取中。最后显示一条提示信息，告诉客户怎么退出程序。

每个仓库项的显示逻辑如下，分为 3 列，基础信息、描述和链接：

func repoText(repo \*Repo) string {

s := "--------------------------------------\n"

s += fmt.Sprintf(`Repo: %s | Language: %s | Stars: %d | Forks: %d | Stars today: %d

`, repo.Name, repo.Lang, repo.Stars, repo.Forks, repo.Add)

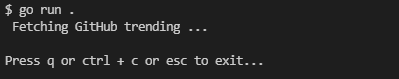
s += fmt.Sprintf("Desc: %s\n", repo.Desc)

s += fmt.Sprintf("Link: %s\n", repo.Link)

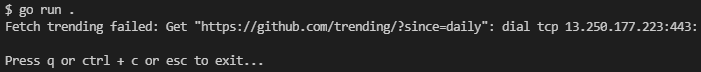
return s

}

运行（多文件运行不能用go run main.go）：



获取失败（国内 GitHub 不稳定，多试几次总会遇到😭）：



获取成功：



让界面更美观

黑白色我们已经看了太多太多了，能不能让字体呈现不同的颜色呢？当然可以。bubbletea可以利用lipgloss库给文本添加各种颜色，我们定义了 4 种颜色，颜色的 RBG 值是我在[http://tool.chinaz.com/tools/pagecolor.aspx](http://tool.chinaz.com/tools/pagecolor.aspx" \t "_blank)挑的：

var (

cyan = lipgloss.NewStyle().Foreground(lipgloss.Color("#00FFFF"))

green = lipgloss.NewStyle().Foreground(lipgloss.Color("#32CD32"))

gray = lipgloss.NewStyle().Foreground(lipgloss.Color("#696969"))

gold = lipgloss.NewStyle().Foreground(lipgloss.Color("#B8860B"))

)

想要将文本变为什么颜色，只需要调用对应颜色对象的Render()方法将文本传入即可。例如我们想让提示变为暗灰色，中间文字使用暗黄色，修改View()方法：

**func** (m model) **View**() **string** {

**var** s **string**

**if** m.err != nil {

s = gold.Render(fmt.Sprintf("fetch trending failed: %v", m.err))

} **else** **if** len(m.repos) > 0 {

**for** \_, repo := **range** m.repos {

s += repoText(repo)

}

s += cyan.Render("--------------------------------------")

} **else** {

s = gold.Render(" Fetching GitHub trending ...")

}

s += "\n\n"

s += gray.Render("Press q or ctrl + c or esc to exit...")

**return** s + "\n"

}

然后仓库的基本信息我们用青色（cyan），描述用绿色，链接用暗灰色：

func repoText(repo \*Repo) string {

s := cyan.Render("--------------------------------------") + "\n"

s += fmt.Sprintf(`Repo: %s | Language: %s | Stars: %s | Forks: %s | Stars today: %s

`, cyan.Render(repo.Name), cyan.Render(repo.Lang), cyan.Render(strconv.Itoa(repo.Stars)),

cyan.Render(strconv.Itoa(repo.Forks)), cyan.Render(strconv.Itoa(repo.Add)))

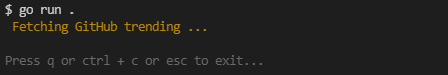
s += fmt.Sprintf("Desc: %s\n", green.Render(repo.Desc))

s += fmt.Sprintf("Link: %s\n", gray.Render(repo.Link))

return s

}

再次运行：



成功：



嗯，现在好看多了。

我没有偷懒

有时候网络很慢，加上一个请求正在处理的提示能让我们更放心（程序还在跑，没偷懒）。bubbletea的兄弟仓库bubbles提供了一个叫做spinner的组件，它只是显示一些字符，一直在变化，给我们造成一种任务正在处理中的感觉。spinner在github.com/charmbracelet/bubbles/spinner包中，需要先引入。然后在模型中增加spinner.Model字段：

**type** model struct {

repos []\***Repo**

err error

spinner spinner.**Model**

}

创建模型时，同时需要初始化spinner.Model对象，我们指定spinner的文本颜色为紫色：

**var** purple = lipgloss.NewStyle().Foreground(lipgloss.Color("#800080"))

func newModel() model {

sp := spinner.NewModel()

sp.Style = purple

return model{

spinner: sp,

}

}

spinner通过Tick来触发其改变状态，所以需要在Init()方法中返回触发Tick的Cmd。但是又需要返回fetchTrending。bubbletea提供了Batch可以将两个Cmd合并在一起返回：

**func** (m model) **Init**() **tea**.**Cmd** {

**return** tea.Batch(

spinner.Tick,

fetchTrending,

)

}

然后Update()方法中我们需要更新spinner。Init()方法返回的spinner.Tick会产生spinner.TickMsg，我们对其做处理：

case spinner.TickMsg:

**var** cmd tea.Cmd

m.spinner, cmd = m.spinner.Update(msg)

return m, cmd

spinner.Update(msg)返回一个tea.Cmd对象驱动下一次Tick。

最后在View()方法中，我们将spinner显示出来。调用其View()方法返回当前状态的字符串，拼在我们想要显示的位置：

**func** (m model) **View**() **string** {

**var** s **string**

**if** m.err != nil {

s = gold.Render(fmt.Sprintf("fetch trending failed: %v", m.err))

} **else** **if** len(m.repos) > 0 {

**for** \_, repo := **range** m.repos {

s += repoText(repo)

}

s += cyan.Render("--------------------------------------")

} **else** {

*// 这里*

s = m.spinner.View() + gold.Render(" Fetching GitHub trending ...")

}

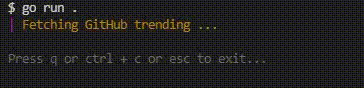
s += "\n\n"

s += gray.Render("Press q or ctrl + c or esc to exit...")

**return** s + "\n"

}

运行：



分页

由于一次返回了很多 GitHub 仓库，我们想对其进行分页显示，每页显示 5 条，可以按pageup和pagedown翻页。首先在模型中增加两个字段，当前页和总页数：

**const** (

CountPerPage = 5

)

**type** model **struct** {

*// ...*

curPage **int**

totalPage **int**

}

拉取到仓库时，计算总页数：

**case** []\*Repo:

m.repos = msg

m.totalPage = (len(msg) + CountPerPage - 1) / CountPerPage

**return** m, nil

另外需要监听翻页按键：

**case** "pgdown":

if m.curPage < m.totalPage-1 {

m.curPage++

}

**return** m, **nil**

**case** "pgup":

if m.curPage > 0 {

m.curPage--

}

**return** m, **nil**

在View()方法中，我们根据当前页计算需要显示哪些仓库：

**start**, **end** := m.curPage\*CountPerPage, (m.curPage+1)\*CountPerPage

**if** **end** > len(m.repos) {

end = len(m.repos)

}

**for** \_, repo := range m.repos[**start**:**end**] {

s += repoText(repo)

}

s += cyan.Render("--------------------------------------")

最后，如果总页数大于 1，给出翻页按键的提示：

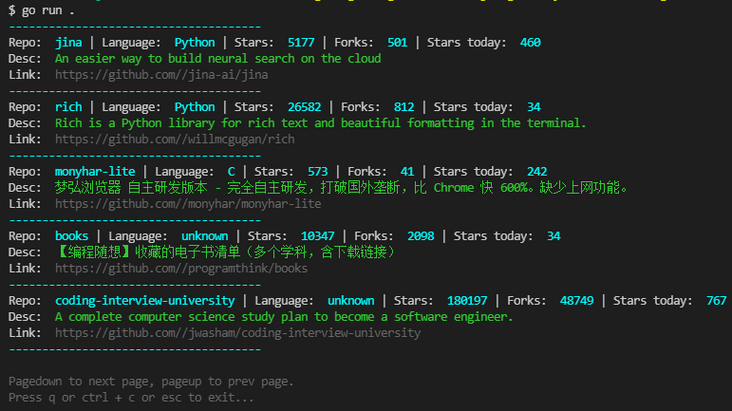
if m.totalPage > 1 {

s += gray.Render("Pagedown to next page, pageup to prev page.")

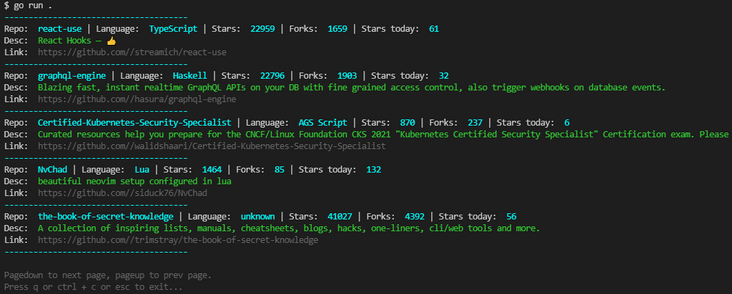
s += "\n"

}

运行：



很棒，我们只显示了 5 页。试试翻页吧：



总结

bubbletea提供了一个 TUI 程序运行的基本框架。我们要显示什么，显示的样式，要对哪些事件进行处理都由我们自己指定。bubbletea仓库的examples文件夹中有多个示例程序，对编写 TUI 程序感兴趣的童鞋千万不能错过。另外它的兄弟仓库bubbles中也提供了不少组件。

大家如果发现好玩、好用的 Go 语言库，欢迎到 Go 每日一库 GitHub 上提交 issue😄