太赞了，使用 go-pretty 轻松美化终端输出

在学习 Go 电子表格操作库 [Excelize](http://cxyroad.com/ ”https://github.com/qax-os/excelize”) 时，对读取的数据在控制台输出结果显示不太满意，在想有没有相关的美化表格输出的开源库，于是搜索一番发现了 [go-pretty](http://cxyroad.com/ ”https://github.com/jedib0t/go-pretty/”) 这个库，试用下来功能还挺强大的，这里记录一下方便日后查阅。

作者在源代码中的 `https://github.com/jedib0t/go-pretty/tree/main/table` 位置列出了库的一些功能点，并给出了部分示例代码，在这里可以初步掌握其用法。此外在源码 `https://github.com/jedib0t/go-pretty/tree/main/cmd` 处有完整的示例代码可供参考。

文章示例以 [Titanic 数据集](http://cxyroad.com/) 作为数据源，并通过 Excel 将其转换成了 XLSX 格式保存在了系统 E 盘中。下面是通过 Excelize 读取数据的代码：

```
package main

import (
”fmt”
”os”

”github.com/jedib0t/go-pretty/v6/table”
”github.com/jedib0t/go-pretty/v6/text”
”github.com/xuri/excelize/v2”
)

func main() {
f, err := excelize.OpenFile(”E:\\ExcelDemo\\titanic.xlsx”)
if err != nil {
fmt.Println(err)
return
}
defer func() {
if err := f.Close(); err != nil {
fmt.Println(err)
}
}()
```

```
rows, err := f.GetRows(”titanic”)
if err != nil {
fmt.Println(err)
return
}

// 后续代码...
}
```

上述代码中我们读取的是 `titanic` 工作簿中的数据，这个是转换保存后的结果，默认情况下我们创建一个空白的工作簿，它的名称是 `Sheet1`。

输出表格数据

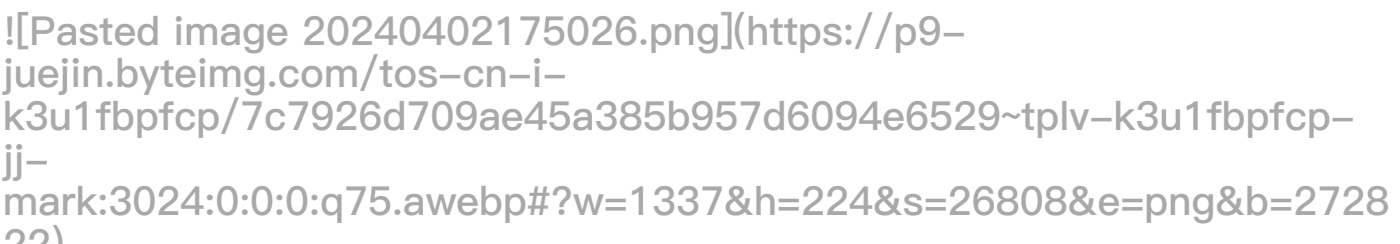
在 Go-pretty 中我们通过调用 `table.NewWriter()` 获得一个 `Table` 实例 `t`，然后调用 `t.AppendHeader()`，`t.AppendRow()`，`t.AppendRows()` 和 `t.AppendFooter()` 来分别设置表格头部，内容和表尾内容，最后调用 `t.Render()` 完成最终的渲染输出。

在 Excelize 中通过 `f.GetRows()` 和 `f.GetCols()` 分别获取行和列的数据，返回的是一个包含数据行或列的切片，我们需要把每一行数据通过 `table.Row()` 包装后传入 `t.AppendXx()` 系列方法中，所以需要遍历每一个切片数据。

```
// 实例化一个表格
t := table.NewWriter()
// 标准输出
t.SetOutputMirror(os.Stdout)

// 获取表头数据
tableHeader := make(table.Row, 0)
for _, cell := range rows[0] {
tableHeader = append(tableHeader, cell)
}
t.AppendHeader(tableHeader)

// 这里只读取前5行数据
for _, row := range rows[1:5] {
innerRow := make(table.Row, 0)
```

```
for _, cell := range row {
innerRow = append(innerRow, cell)
}
t.AppendRow(innerRow)
t.AppendSeparator()
// 这里不用担心最后一行数据的分隔线会多出一行，不需要额外的判断
// if index != len(rows[1:]) {
// t.AppendSeparator()
// }
}
// 设置表尾
t.AppendFooter(table.Row{"乘客ID", "获救情况", "乘客等级", "姓名", "性别", "年龄", "堂兄弟妹个数", "父母与小孩个数", "船票信息", "票价", "船舱", "登船的港口"})

t.Render()
```

结果：

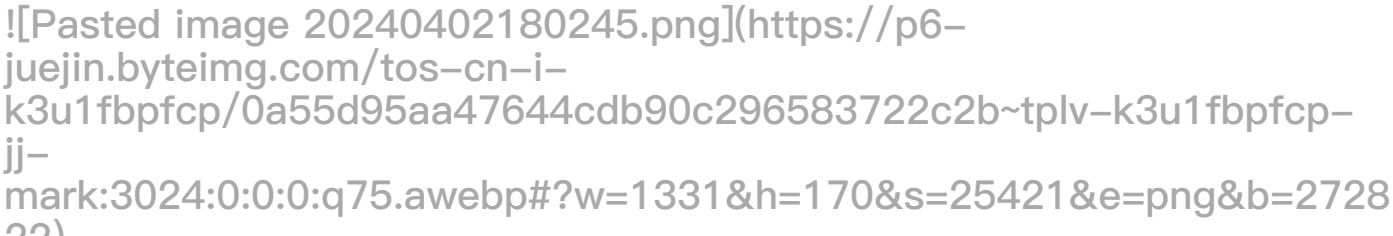


上面的代码中我们使用 `t.AppendRow()` 来添加数据，并在每行数据后面加了一个分隔线，当然我们也可以使用 `t.AppendRows()` 来添加分组数据，遍历方式稍有不同：

```
// 实例化一个表格
t := table.NewWriter()
// 标准输出
t.SetOutputMirror(os.Stdout)

// 获取表头数据
tableHeader := make(table.Row, 0)
for _, cell := range rows[0] {
tableHeader = append(tableHeader, cell)
}
t.AppendHeader(tableHeader)
```

```
tableBody := make([]table.Row, 0)
for _, row := range rows[1:5] {
innerRow := make(table.Row, 0)
for _, cell := range row {
innerRow = append(innerRow, cell)
}
tableBody = append(tableBody, innerRow)
}
// 设置表格内容
t.AppendRows(tableBody)
// 设置表尾
t.AppendFooter(table.Row{"乘客ID", "获救情况", "乘客等级", "姓名", "性别", "年龄", "堂兄弟妹个数", "父母与小孩个数", "船票信息", "票价", "船舱", "登船的港口"})

t.Render()
```

结果：



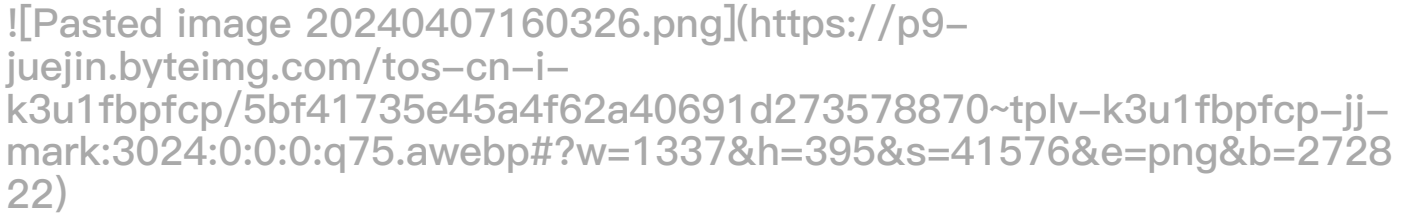
自动合并单元格

通过配置 `RowConfig` 和 `ColumnConfig` 可以实现自动在水平和垂直合并指定的单元格，并设置其合并后的数据对齐方式。**这个功能只适合使用 `t.Render()` 渲染输出，对于 `CSV/HTML/Markdown/TSV` 模式输出并不支持。**

```
rowConfigAutoMerge := table.RowConfig{AutoMerge: true}
t := table.NewWriter()
t.SetOutputMirror(os.Stdout)

tableHeader := make(table.Row, 0)
for _, cell := range rows[0] {
```

```
tableHeader = append(tableHeader, cell)
}
t.AppendHeader(tableHeader, rowConfigAutoMerge)

for _, row := range rows[1:10] {
innerRow := make(table.Row, 0)
for _, cell := range row {
innerRow = append(innerRow, cell)
}
t.AppendRow(innerRow, rowConfigAutoMerge)
t.AppendSeparator()
}
t.AppendFooter(table.Row{"乘客ID", "获救情况", "乘客等级", "姓名", "性别", "年龄", "堂兄弟妹个数", "父母与小孩个数", "船票信息", "票价", "船舱", "登船的港口"})
t.SetColumnConfigs([]table.ColumnConfig{
// 第2列进行合并
{Number: 2, AutoMerge: true},
})
t.Render()
```

结果：

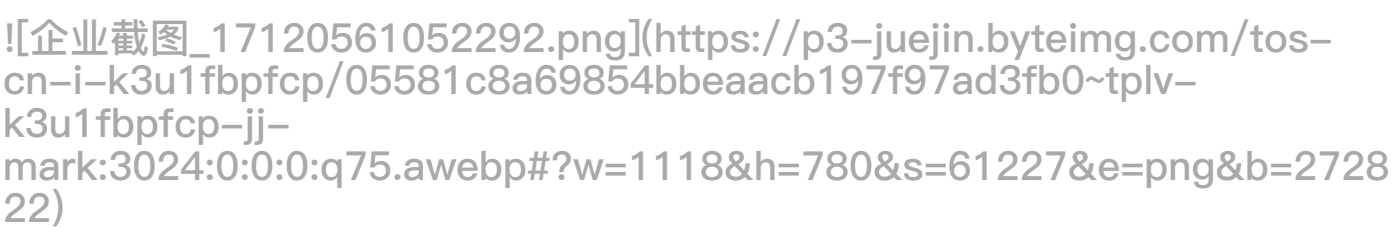


表格的行和列都可以单独或者同时设置对齐方式，水平为 `Align`（示例：`Align: text.AlignCenter`），垂直为 `VAlign`（示例：`VAlign: VAlignMiddle`）。由于输出的是纯文本格式，因此在垂直方向是做不到绝对居中的，我们通过修改数据源列 `NAME` 的其中一行数据使其文本内容增加以显示出多行，然后限制这一行的宽度来查看实际效果，发现只有奇数行居中了，这是符合实际的。但是合并后的单元格并没有跨行居中，只是在其所在单元格同样以奇数行居中。

```
t.SetColumnConfigs([]table.ColumnConfig{
{
Number:    2,
AutoMerge: true,
```

```
VAlign:    text.VAlignMiddle,
Align:     text.AlignCenter,
},
{
Number: 3,
VAlign: text.VAlignMiddle,
Align:  text.AlignCenter,
},
{
Number:   4,
WidthMax: 20,
},
})

```

结果：



> 注意：在 `ColumnConfig` 中我们是通过 `Number` 来指定要配置的列，也可以通过 `Name` 来设置。如果
> 同时设置，前都会覆盖后者。对于 `Name` 配置需要注意一点的是输出结果中的表头显示是被格式化成全部大写了，这里我们需要指定原始的形式，比如最后一行应为：`Embarked`。

> 此外：在内容为中英文加符合的情况下，输出结果也会出现不对齐的情况。

上面我们对内容进行了水平和垂直对齐设置，对于表头和表尾我们可以通过 `AlignHeader` 和 `AlignFooter` 来分别设置每一列的对齐。

水平对齐方式可选值：`AlignDefault`, `AlignLeft`, `AlignCenter`, `AlignJustify`, `AlignRight`, `AlignAuto`，默认为：`AlignLeft`, 自动情况下，数字向右对齐，其它情况向左对齐。

垂直对齐方式可选值：`VAlignDefault`, `VAlignTop`, `VAlignMiddle` 和

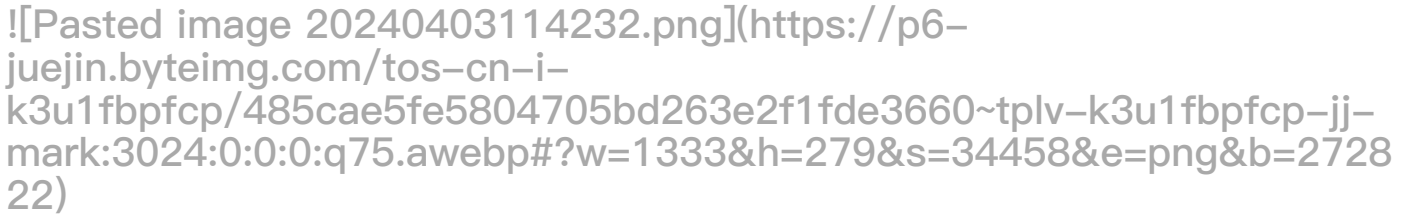
`VAlignBottom`，默认为：`VAlignTop`。

添加表格标题和描述

通过调用 `SetTitle()` 和 `SetCaption()` 来设置标题和描述文字，渲染后标题位于表头上方一行，不可设置对齐方式（默认左对齐），而描述则位于表格的下方。由于这 2 个方法实际是对 `fmt.Printf()` 的一个透明封装，因此我们可以传多个参数来格式化输出。

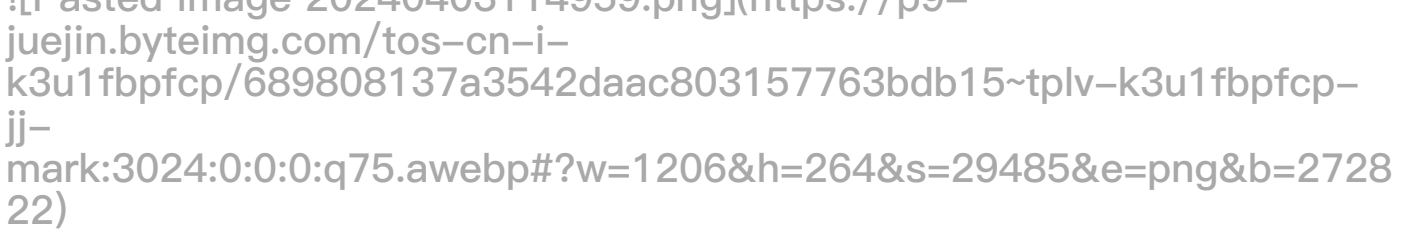
```
t.SetTitle("泰坦尼克号数据集")
t.SetCaption("%s 数据集作为 kaggle 比赛中的经典数据集，非常适合作为数据分析入门的练手数据，同时网上也有许多分析处理案例。", "Titanic")
```

结果：



自动索引

通过调用 `t.SetAutoIndex(true)` 来激活自动索引功能，我们取数据第 10-15 条，显示结果如下：

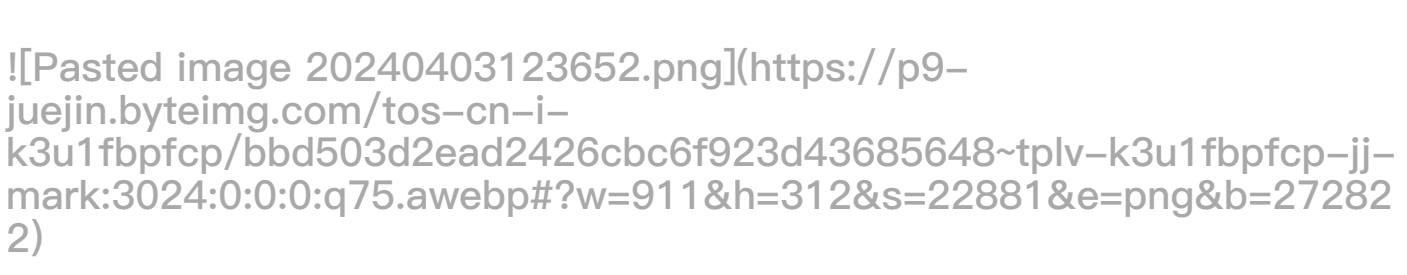


限制行宽度和列宽度

如果行的内容过长的话，只会换行，不会出现水平滚动条，因此 Go-pretty 提供了 `t.SetAllowedRowLength()` 方法来限制宽度，超出的部分在每行以 `~` 来显示。对于列的限制我们只需要在 `table.ColumnConfig` 中设置 `MinWidth` 和 `MaxWidth` 即可，内容会自动换行。

```
t.SetAllowedRowLength(120)
t.SetColumnConfigs([]table.ColumnConfig{
{Number: 4, WidthMax: 16},
})
```

结果：



这里需要注意的是默认的换行方式为 `text.WrapText`，如果需要根据内容选择不同的换行方式（官方支持三种 `WrapHard/WrapSoft/WrapText`），可以在 `WidthMaxEnforcer` 配置中进行判断和指定，具体如何选用需要根据文字内容显示的结果合理选用。

```
t.SetColumnConfigs([]table.ColumnConfig{
{
Number:   4,
WidthMax: 16,
WidthMaxEnforcer: func(col string, maxLen int) string {
// 根据 col 和 maxLen 来选择合适的换行方式
return text.WrapText(col, maxLen)
},
},
})
```

三种换行方式对比如下：

![Pasted image 20240403123112.png](https://p1-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/e1a8eac2f21141babedb13e45905fb1e~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=807&h=339&s=75680&e=png&b=ffffff)

分页显示

对于数据较多的场景，可以使用分页功能，按指定的大小，来分页显示，只需要设置 `t.SetPageSize()` 即可。

![Pasted image 20240408121509.png](https://p1-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/bf3c0b79625f4f33aa1941cbe61b4190~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1334&h=595&s=65226&e=png&b=272822)

数据排序

我们可以按指定的列进行排序，比如先按年龄排序，然后相同年龄再按票价进行排序。这里需要使用到 `t.SortBy()` 方法。排序方式有：`Asc`, `AscAlphaNumeric`, `AscNumeric`, `AscNumericAlpha`, `Dsc`, `DscAlphaNumeric`, `DscNumeric` 和 `DscNumericAlpha` 8 种方式。

```
t.SortBy([]table.SortBy{
{Name: ”Age”, Mode: table.Asc},
{Name: ”Fare”, Mode: table.AscNumeric},
})
```

排序后的结果：

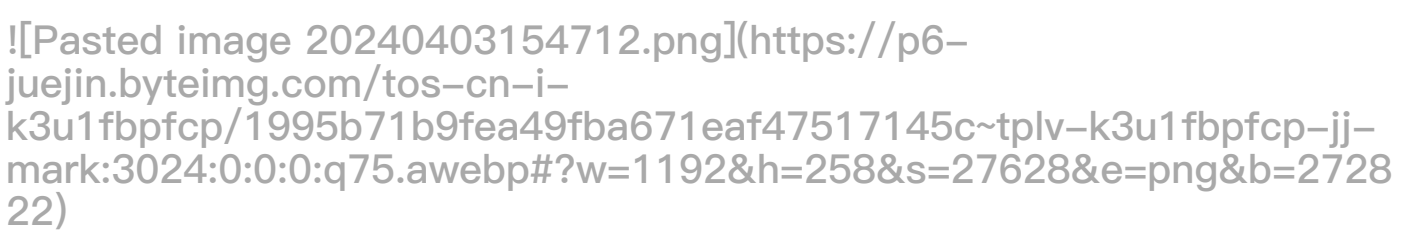
![Pasted image 20240403152635.png](https://p9-juejin.byteimg.com/tos-cn-i-k3u1fbpfcp/1e32b9dbe1c545c7880ec3fbe2b05f5a~tplv-k3u1fbpfcp-jj-mark:3024:0:0:0:q75.awebp#?w=1188&h=258&s=24656&e=png&b=272822)

设置单元格颜色

在 `t.SetColumnConfigs()` 中配置 `table.ColumnConfig` 的 `Colors`, `ColorsHeader`, `ColorsFooter` 可以实现对表头、内容和表尾前景后和背景设置。

```
t.SetColumnConfigs([]table.ColumnConfig{
{
Name: ”Pclass”,
Colors: text.Colors{
text.FgHiBlue,
},
ColorsHeader: text.Colors{
text.BgCyan,
},
},
{
Name: ”Sex”,
Colors: text.Colors{
text.BgHiGreen,
text.FgHiYellow,
},
},
})
```

结果如下：



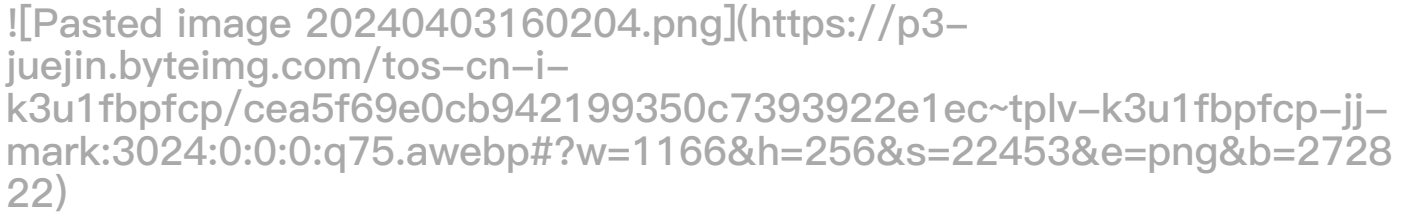
单元格格式化

很多场景下我们需要对数据作转换，根据条件以不同的方式来显示数据，对脏数据设置默认值等。我们可以通过 `Transformer`, `TransformerFooter` 和 `TransformerHeader` 来对表头、内容和表尾进行转换。下面我们对性别用颜

色加图标替换显示：

```
t.SetColumnConfigs([]table.ColumnConfig{
{
Name: ”Sex”,
Transformer: func(val interface{}) string {
if val == ”male” {
return text.Colors{text.FgBlue}.Sprintf(”♂”)
} else if val == ”female” {
return text.Colors{text.FgRed}.Sprintf(”♀”)
} else {
return ””
}
},
},
})
```

结果如下：

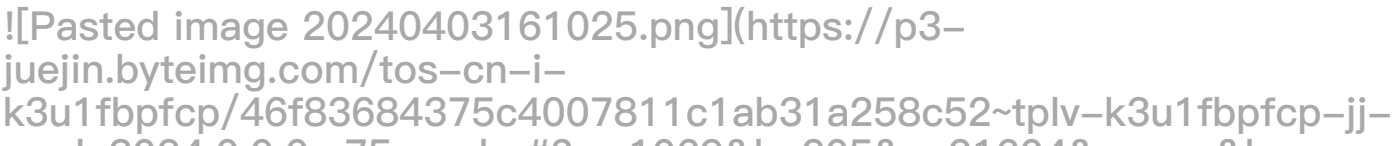


切换主题

Go-pretty 默认提供了很多主题和不同的边框类型。由于文章最开始提及的官方文档中有很详细的说明，这里就不再赘述了，而且很多时候我们并不需要，直接上效果图：

```
t.SetStyle(table.StyleColoredBright)
```

结果如下：



输出不同的文档格式

上面所有的示例都是在控制台中输出格式化后的表格数据，我们也可以调用相应的函数格式化输出为 CSV/TSV/HTML Table/Markdown Table 的表格描述方式。

* `t.RenderCSV()` 输出 CSV，以逗号分隔数据的纯文本
* `t.RenderTSV()` 输出 TSV，即以制表符分隔数据的纯文本
* `t.RenderHTML()` 输出 HTML 表格表示
* `t.RenderMarkdown()` 输出 Markdown 表格表示

注意：单元格合并仅支持通用输出 `t.Render()`。

输出列表数据

`go-pretty/list` 提供了数据列表格式化输出的一些功能，就像我们在 HTML 中使用 `<ul><li>fqgx</li></ul>` 一样，当然我们也可以直接渲染输出为 HTML 表示。

要想生成一个列表，可以通过 `l := list.NewWriter()` 或者 `l := list.List{}` 直接实例化来初始化。然后就可以通过 `l.AppendItem()` 来添加列表项数据，使用 `l.Indent()` / `l.UnIndent()` / `UnIndentAll()` 来增加/减少/重置所有缩进。

默认样式

默认输出样式其实和 HTML 中的 `<ul>` 是一样的。

```
package main
```

```
import (
”fmt”

”github.com/jedib0t/go-pretty/v6/list”
)

func main() {
l := list.List{}
l.AppendItem(”Item 1”)
l.AppendItem(”Item 2”)
l.AppendItem(”Item 3”)
l.Indent()
l.AppendItem(”Item 4”)
l.AppendItem(”Item 5”)
l.UnIndent()
l.AppendItem(”Item 6”)
l.AppendItem(”Item 7”)
fmt.Println(l.Render())
// fmt.Println(list.RenderHTML())
}

// 结果:
// * Item 1
// * Item 2
// * Item 3
//   * Item 4
//   * Item 5
// * Item 6
// * Item 7
```

其它可选样式

除了默认的效果外，官方还提供了多种候选样式。

```
l.SetStyle(list.StyleBulletCircle)
// ● Item 1
// ● Item 2
// ● Item 3
//   ● Item 4
//   ● Item 5
// ● Item 6
```

```
// ● Item 7

l.SetStyle(list.StyleBulletFlower)
//  Item 1
//  Item 2
//  Item 3
//    Item 4
//    Item 5
//  Item 6

l.SetStyle(list.StyleBulletSquare)
// ■ Item 1
// ■ Item 2
// ■ Item 3
//   ■ Item 4
//   ■ Item 5
// ■ Item 6
// ■ Item 7

l.SetStyle(list.StyleBulletStar)
// ★ Item 1
// ★ Item 2
// ★ Item 3
//   ★ Item 4
//   ★ Item 5
// ★ Item 6
// ★ Item 7

l.SetStyle(list.StyleConnectedBold)
//  ┏━ Item 1
//  ┣━ Item 2
//  ┣━ Item 3
//  ┃   ┣━ Item 4
//  ┃   ┗━ Item 5
//  ┣━ Item 6
//  ┗━ Item 7

l.SetStyle(list.StyleConnectedDouble)
//  ╔═ Item 1
//  ╠═ Item 2
//  ╠═ Item 3
//  ║   ╠═ Item 4
//  ║   ╚═ Item 5
//  ╠═ Item 6
//  ╚═ Item 7

l.SetStyle(list.StyleConnectedLight)
//  ┌─ Item 1
```

```
//  ├── Item 2
//  ├── Item 3
//  │    ├── Item 4
//  │    └── Item 5
//  ├── Item 6
//  └── Item 7

l.SetStyle(list.StyleConnectedRounded)
//  ┌── Item 1
//  ├── Item 2
//  ├── Item 3
//  │    ├── Item 4
//  │    └── Item 5
//  ├── Item 6
//  └── Item 7

l.SetStyle(list.StyleMarkdown)
// * Item 1
// * Item 2
// * Item 3
//    * Item 4
//    * Item 5
// * Item 6
// * Item 7

```

自定义样式

如果默认的样式可能满足不了你的需求，作者还提供了灵活的自定义设置。

下面是一个简单的示例：

```
package main

import (
"fmt"

"github.com/jedib0t/go-pretty/v6/list"
"github.com/jedib0t/go-pretty/v6/text"
)

func main() {
```

```
l := list.List{}
funkyStyle := list.Style{
CharItemSingle:    "",
CharItemTop:       "",
CharItemFirst:     "",
CharItemMiddle:    "",
CharItemVertical: "  ",
CharItemBottom:    "",
CharNewline:       "\n",
Format:            text.FormatUpper,
LinePrefix:        "",
Name:              "styleTest",
}
l.SetStyle(funkyStyle)
l.AppendItem("Item 1")
l.AppendItem("Item 2")
l.AppendItem("Item 3")
l.Indent()
l.AppendItem("Item 4")
l.AppendItem("Item 5")
l.UnIndent()
l.AppendItem("Item 6")
l.AppendItem("Item 7")
fmt.Println(l.Render())
}

//  ITEM 1
//  ITEM 2
//  ITEM 3
//    ITEM 4
//    ITEM 5
//  ITEM 6
//  ITEM 7

```

上述代码中我们使用 `text.FormatUpper` 来将选项名称格式化成大写字母，可选择的值还有

* `FormatLower` – 全部小写
* `FormatTitle` – 标题首字母大宇
* `FormatDefault` – 默认效果，不格式化

其它渲染方式

除了默认的渲染方式 `l.Render()`，还支持输出为 HTML，使用 `l.RenderHTML()` 以及 Markdown, 使用 `l.RenderMarkdown()`。在输出 HTML 时，我们还可以使用 `l.SetHTMLCSSClass("ul")` 来指定 `<ul>` 元素的样式。

```
package main

import (
"fmt"

"github.com/jedib0t/go-pretty/v6/list"
)

func main() {
l := list.List{}
l.AppendItem("Item 1")
l.AppendItem("Item 2")
l.AppendItem("Item 3")
l.Indent()
l.AppendItem("Item 4")
l.AppendItem("Item 5")
l.UnIndent()
l.AppendItem("Item 6")
l.AppendItem("Item 7")
l.SetHTMLCSSClass("ul")
fmt.Println(l.RenderHTML())
}

// <ul class="ul">
//   <li>Item 1</li>
//   <li>Item 2</li>
//   <li>Item 3</li>
//   <ul class="ul-1">
//     <li>Item 4</li>
//     <li>Item 5</li>
//   </ul>
//   <li>Item 6</li>
//   <li>Item 7</li>
// </ul>
```

进度条

Go 社区中有很多进度条相关的库，例如：

* [schollz/progressbar: A really basic thread-safe progress bar for Golang applications (github.com)](http://cxyroad.com/”https://github.com/schollz/progressbar”)
* [vbauerster/mpb: multi progress bar for Go cli applications (github.com)](http://cxyroad.com/”https://github.com/vbauerster/mpb”)
* [cheggaaa/pb: Console progress bar for Golang (github.com)](http://cxyroad.com/ ”https://github.com/cheggaaa/pb”)

显然没有使用过以上列出的 3 个库，但是从其官方介绍来看和 Go-pretty 提供的进度条相对比在功能上还是相对较简单，但是 Go-pretty 的作者并没有给出一个入门使用文档，而是在源文件中给出了一个复杂的示例，初看时会一头雾水，理解后会发现这个进度条真的很美观，很强大。

作者介绍其进度条有以下功能点：

* 同时跟踪一个或多个任务
* 在 `Render()` 过程中动态添加一个或多个任务跟踪器
* 当没有更多的跟踪器时，选择让 Writer 自动停止渲染或者手动使用 `stop()` 停止
* 将输出重定向到 `io.Writer` 对象 (如 `os.StdOut`)
* 完全可自定义的样式
* 许多现成的样式
* 使用 `StyleColors` 为跟踪器的各个部分着色
* 使用 `StyleOptions` 自定义跟踪器的渲染方式

实际效果推荐看官方仓库中的 Gif 图。

进度条效果非常适合来演示文件下载的进度，下面给出一个示例：下载指定 URL 的资源，创建临时目录存放下载的资源，资源下载完成后删除临时目录并存放至指定目录中。

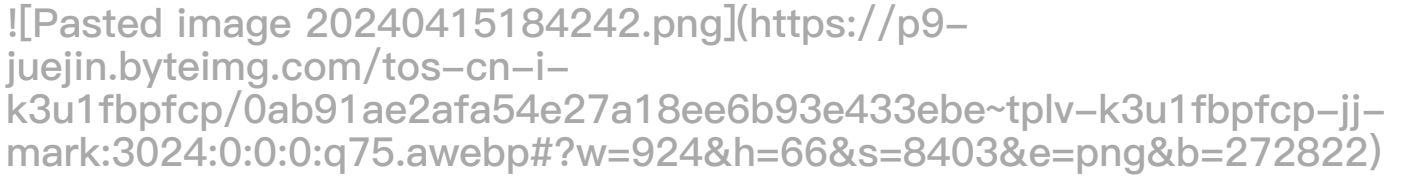
> 作者 Go 新手：代码就不作封装了...

```
package main

import (
”fmt”
”io”
”net/http”
”net/url”
”os”
”path/filepath”
”time”

”github.com/jedib0t/go-pretty/v6/progress”
”github.com/jedib0t/go-pretty/v6/text”
tsize ”github.com/kopoli/go-terminal-size”
)

type Downloader struct {
Total    int64  // 文件大小
Current  int64  // 当前接收的字节数
Last     int64  // 上一次接收的字节数
filename string // 文件名
pw       progress.Writer
tracker  *progress.Tracker
}

func (d *Downloader) Write(p []byte) (int, error) {
n := len(p)
d.Current += int64(n)
increment := d.Current - d.Last
d.Last = d.Current

d.tracker.Increment(increment)

if d.Current == d.Total {
d.tracker.Total = 0
d.tracker.MarkAsDone()
}

return n, nil
}

func DownloadFile(imgPath string, url string) error {
// 创建目录
tempDir, err := os.MkdirTemp(”.”, ”dir”)
if err != nil {
return err
}
```

```
defer os.Remove(tempDir)

// 创建临时文件
tmpFile, err := os.CreateTemp(tempDir, imgPath)
if err != nil {
return err
}
defer tmpFile.Close()
defer os.Remove(tmpFile.Name())

// 下载文件
resp, err := http.Get(url)
if err != nil {
return err
}
defer resp.Body.Close()

counter := &Downloader{}
counter.Total = resp.ContentLength
counter.Last = 0
counter.filename = imgPath

width := 100
if size, err := tsize.GetSize(); err == nil {
width = size.Width
}

counter.pw = progress.NewWriter()
counter.pw.SetTrackerLength(width / 3)
counter.pw.SetMessageLength(width / 2)
counter.pw.SetAutoStop(true)
counter.pw.SetStyle(progress.StyleDefault)
counter.pw.SetUpdateFrequency(time.Millisecond * 100)
counter.pw.Style().Colors = progress.StyleColorsExample
counter.pw.Style().Options.DoneString = text.FgBlue.Sprint("下载完成")
counter.pw.Style().Options.ErrorString = text.FgRed.Sprint("下载失败")
counter.pw.Style().Options.PercentFormat = "%4.1f%%"
counter.pw.Style().Visibility.ETA = true
counter.pw.Style().Visibility.Pinned = true

go counter.pw.Render()

message := fmt.Sprintf("Downloading %s", counter.filename)
units := progress.UnitsBytes
tracker := progress.Tracker{
Total:      counter.Total,
Message:    message,
Units:      units,
```

```
DeferStart: false,
}
counter.pw.AppendTracker(&tracker)
counter.tracker = &tracker

if _, err := io.Copy(tmpFile, io.TeeReader(resp.Body, counter)); err != nil {
tmpFile.Close()
os.Remove(tmpFile.Name())
return err
}

for counter.pw.IsRenderInProgress() {
if counter.pw.LengthActive() == 0 {
counter.pw.Stop()
}
}

// 确保临时文件的内容已经刷新到磁盘
err = tmpFile.Sync()
if err != nil {
panic(err)
}

tmpFile.Close()

// 将临时文件重命名为最终文件
if err = os.Rename(tmpFile.Name(), filepath.Join("image", imgPath)); err
!= nil {
return err
}
return nil
}

func main() {
fmt.Println("Download Started")

fileUrl := "http://212.183.159.230/512MB.zip"
parsedUrl, err := url.Parse(fileUrl)
if err != nil {
panic(err)
}
path := filepath.Base(parsedUrl.Path)
err = DownloadFile(path, fileUrl)
if err != nil {
panic(err)
}

fmt.Println("Download Finished")
```

}

```

效果截图：




建议仔细研究一下官方关于进度条的示例，然后再去 Github 上看看别人怎么封装使用的，作者本身就是初学者，所以没有相关项目经验，只能初窥门径，能简单用起来，作个 Demo 效果。

后续随着学习的深入和经验的积累，再分享具体项目中使用。

写在最后
---

感谢库作者 [jedib0t](http://cxyroad.com/ "https://github.com/jedib0t")，在写作文章时发现了一些 Bug，作者很快就回复修复了。

* [Paging result not expected · Issue #312 · jedib0t/go-pretty (github.com)](http://cxyroad.com/ "https://github.com/jedib0t/go-pretty/issues/312")
* [Paging is set when row/column cell merge is set at the same time, missing some cell data · Issue #315 · jedib0t/go-pretty (github.com)](http://cxyroad.com/ "https://github.com/jedib0t/go-pretty/issues/315")

文章参考：
---

* [【Go】用Go在命令行输出好看的表格\_go-pretty-CSDN博客](http://cxyroad.com/ "https://blog.csdn.net/Meepoljd/article/details/129422612")
* 编程日记:分享开源库go-pretty - 掘金 (juejin.cn)
  原文链接: https://juejin.cn/post/7357921175559077929
```