使用Angular组件定制ag-Grid

文章：

https://blog.ag-grid.com/learn-to-customize-angular-grid-in-less-than-10-minutes/?\_ga=2.73943506.1707452238.1578881683-1066262163.1578881683

源代码：

<https://github.com/angular-grid/customization-demo>

客户喜欢我们的Angular datagrid组件，因为它有大量的定制选项。在ag-Grid中，我们认为开发人员应该能够轻松地定制网格并实现功能来满足他们的业务需求。由于基于组件的体系结构，ag-Grid非常灵活。

您可以通过创建新的Angular组件并将它们集成到datagrid中来扩展默认的功能。

在前一篇关于如何开始使用Angular grid的文章中，我们已经将ag-Grid集成到了Angular应用程序中。如果您还没有完成我们之前的教程，也没有样例应用程序，那么您可以从这个GitHub存储库下载它。

**自定义渲染器**

首先，我们将实现一个自定义单元格呈现器，以显示根据用户的区域设置格式化的价格。下面是我电脑上显示的数字:

**自定义单元格编辑器**  
因为“Price”字段是数字的，所以它不能包含非数字字符。默认情况下，当一个单元格被编辑时，用户可以输入任何字符。因此，我们将实现一个自定义单元格编辑器，以限制输入只有数字:

**自定义过滤器**  
我们将通过实现一个自定义过滤器来完成本教程。我们的新过滤器将提供用户界面，键入价格范围，并过滤掉汽车的范围:

**定制组件**  
我们的Angular网格是通过实现自定义Angular组件来定制的。UI部分是使用Angular模板实现的。还有一些方法需要由组件实现来使用ag-Grid。例如，单元格编辑器应该实现用于从组件请求值并更新数据的getValue方法。自定义列过滤器应该实现处理值等的doesFilterPass方法。这些必需的方法由文档中描述的接口定义。

AgGridModule中应该列出ag-Grid使用的所有自定义组件。将组件方法导入主应用程序模块。我们将实现三个自定义组件，并像这样添加它们:

@NgModule({

imports: [

BrowserModule,

AgGridModule.withComponents([

NumberFormatterComponent,

NumericEditorComponent,

RangeFilterComponent]

)

],

...

})

export class AppModule {}

**自定义渲染器**  
网格的工作是布置单元。默认情况下，网格将使用简单的文本创建单元格值。如果希望在单元格中包含更复杂的HTML，可以使用单元格呈现器实现。我们希望根据用户的地区设置数字的格式，因此需要创建一个自定义组件。组件将利用Angular的货币管道来格式化值。

1 - 定义自定义组件  
第一个任务是实现Angular组件。它将通过agInit方法接收单元格的值。

为了显示格式化的值，我们将使用一个简单的span元素。因此，创建一个新的文件number-formatter.component.js，并把我们的组件的代码放在里面:

@Component({

selector: 'app-number-formatter-cell',

template: `

<span>{{params.value | currency:'EUR'}}</span>

`

})

export class NumberFormatterComponent {

params: any;

agInit(params: any): void {

this.params = params;

}

}

它是一个简单的组件，只有一个用于从ag-Grid接收参数的方法agInit。

2 - 注册组件  
现在我们有了自己的组件，我们需要告诉ag-Grid它。

所有自定义组件都应该在frameworkComponents配置选项中列出。让我们导入自定义单元格渲染器并在配置中指定它:

export class AppComponent implements OnInit {

frameworkComponents = {

numberFormatterComponent: NumberFormatterComponent,

};

}

3 - 指定列的呈现器  
我们几乎完成了。剩下的惟一事情是为“Price”列指定我们的组件作为单元格呈现程序。

我们可以在列定义中这样做:

export class AppComponent implements OnInit {

columnDefs = [

{headerName: 'Make', field: 'make'},

{headerName: 'Model', field: 'model'},

{

headerName: 'Price',

field: 'price',

editable: true,

/\* specify custom cell renderer \*/

cellRenderer: 'numberFormatterComponent',

}

];

现在，如果你运行应用程序，你应该看到价格格式:

**自定义单元格编辑器**  
我们的Angular datagrid提供了丰富的编辑功能。它类似于一个电子表格，允许您以内联方式编辑数据。只需按F2或双击一个单元格，ag-Grid就会激活编辑模式。您不需要为简单的字符串编辑提供自定义编辑器。但是，当我们需要实现自定义逻辑时，比如将输入限制为数字，我们需要创建单元格编辑器。

启用编辑  
在我们开始工作的自定义组件将作为一个编辑器，我们需要启用编辑的价格列:

export class AppComponent implements OnInit {

columnDefs = [

{headerName: 'Make', field: 'make'},

{headerName: 'Model', field: 'model'},

{

headerName: 'Price',

field: 'price',

/\* enable editing \*/

editable: true,

}

];

定义自定义组件  
与自定义单元格呈现器类似，我们需要定义一个Angular组件来充当单元格编辑器。让我们从基本实现开始，它将呈现一个输入元素，当用户激活编辑模式时将弹出:

@Component({

selector: 'app-numeric-editor-cell',

template: `

<input/>

`

})

export class NumericEditorComponent AfterViewInit {}

现在我们需要实现一些必需的方法。

一旦编辑完成按回车或从输入中删除焦点，ag-Grid需要从编辑器中获取一个值。为此，它调用Angular组件上的getValue方法，该方法应该返回编辑的结果。

我们希望返回用户输入的值。所以我们需要访问那个输入来读取它的值。在Angular中，我们可以使用@ViewChild机制来实现:

@Component({

selector: 'app-numeric-editor-cell',

template: `

<input #i [value]="params.value"/>

`

})

export class NumericEditorComponent {

@ViewChild('i') textInput;

params;

agInit(params: any): void {

this.params = params;

}

getValue() {

return this.textInput.nativeElement.value;

}

}

注意，我们在agInit方法中从网格获取单元格值，并在getValue方法中返回该值。

现在我们准备实现过滤非数字字符的功能。为此，我们需要向输入添加一个事件监听器，并检查用户输入的每个项。我们还需要添加keyDown侦听器，以防止用户按下导航键时失去焦点。

我们是这样做的:

@Component({

selector: 'app-numeric-editor-cell',

template: `

<input (keypress)="onKeyPress($event)" (keydown)="onKeyDown($event)"/>

`

})

export class NumericEditorComponent {

...

onKeyPress(event) {

if (!isNumeric(event)) {

event.preventDefault();

}

function isNumeric(ev) {

return /\d/.test(ev.key);

}

}

onKeyDown(event) {

if (event.keyCode === 39 || event.keyCode === 37) {

event.stopPropagation();

}

}

}

好了，现在差不多都准备好了。我们需要做的最后一件事是，一旦用户激活编辑模式，就将焦点设置为自定义单元格编辑器中的输入。为了方便起见，我们可以使用Angular提供的ngAfterViewInit生命周期钩子:

export class NumericEditorComponent implements AfterViewInit {

@ViewChild('i') textInput;

...

ngAfterViewInit() {

setTimeout(() => {

this.textInput.nativeElement.focus();

});

}

}

这里是你复制/粘贴它的完整代码:

@Component({

selector: 'app-numeric-editor-cell',

template: `

<input #i [value]="params.value" (keypress)="onKeyPress($event)" (keydown)="onKeyDown($event)"/>

`

})

export class NumericEditorComponent implements AfterViewInit {

@ViewChild('i') textInput;

params;

ngAfterViewInit() {

setTimeout(() => {

this.textInput.nativeElement.focus();

});

}

agInit(params: any): void {

this.params = params;

}

getValue() {

return this.textInput.nativeElement.value;

}

onKeyPress(event) {

if (!isNumeric(event)) {

event.preventDefault();

}

function isNumeric(ev) {

return /\d/.test(ev.key);

}

}

onKeyDown(event) {

if (event.keyCode === 39 || event.keyCode === 37) {

event.stopPropagation();

}

}

}

注册组件  
与自定义单元格渲染器类似，我们需要在frameworkComponents中注册单元格编辑器，并将其指定为“Price”列的编辑器:

export class AppComponent implements OnInit {

columnDefs = [

{headerName: 'Make', field: 'make'},

{headerName: 'Model', field: 'model'},

{

headerName: 'Price',

field: 'price',

editable: true,

/\* custom cell editor \*/

cellEditor: 'numericEditorComponent',

}

];

frameworkComponents = {

/\* custom cell editor component\*/

numericEditorComponent: NumericEditorComponent,

};

就是这样。现在，我们可以开始执行实现自定义过滤器的最后一个任务了。

**自定义列过滤器**

过滤是数据网格最有用的特性之一。它允许用户放大一组特定的记录。我们的Angular网格提供了一个简单的字符串过滤功能。当需要自定义筛选器类型时，可以使用自定义筛选器组件轻松实现逻辑。我们将根据价格范围对汽车进行过滤。

1 - 使过滤  
在我们开始工作之前，我们的自定义过滤器组件，我们需要在网格启用过滤:

<ag-grid-angular

[enableFilter]="true"

[rowData]="rowData"

[columnDefs]="columnDefs"

[frameworkComponents]="frameworkComponents"

>

</ag-grid-angular>

2 - 定义自定义组件

为了实现自定义过滤器，我们遵循熟悉的自定义方法并定义Angular组件。我们的过滤器将被渲染为一个输入，带有一个按钮来应用过滤器。让我们把这个HTML组件的模板:

@Component({

selector: 'app-range-filter-cell',

template: `

<form>

<input name="filter"/>

<button>Apply</button>

</form>

`

})

export class RangeFilterComponent {}

我们的过滤器允许用户指定一个范围的汽车价格的形式的下限-上限:

我们将定义filter属性以将当前筛选条件存储在组件的状态中。在显示输入时，我们希望用组件状态中的当前筛选条件预填充它。我们将使用值输入绑定来实现这一点。下面是实现这个的代码:

@Component({

selector: 'app-range-filter-cell',

template: `

<form>

<input name="filter" [value]="filter"/>

<button>Apply</button>

</form>

`

})

export class RangeFilterComponent implements AfterViewInit {

filter = '';

}

然后我们需要处理用户输入并将其保存到组件的状态。为此，我们将在表单上注册一个事件监听器，并在表单提交时使用Apply按钮处理输入:

@Component({

selector: 'app-range-filter-cell',

template: `

<form (submit)="onSubmit($event)">

<input name="filter" [value]="filter"/>

<button>Apply</button>

</form>

`

})

export class RangeFilterComponent {

filter = '';

onSubmit(event) {

event.preventDefault();

const filter = event.target.elements.filter.value;

if (this.filter !== filter) {

this.filter = filter;

}

}

}

每当过滤条件发生更改时，我们不仅需要更新状态，而且还需要将更改通知给ag-Grid。我们可以通过agInit方法中的参数调用ag-Grid为我们提供的filterChangedCallback来实现这一点。

让我们添加agInit方法并修改onSubmit处理程序来通知网格:

export class RangeFilterComponent {

filter = '';

params: any;

agInit(params: any): void {

this.params = params;

}

onSubmit(event) {

event.preventDefault();

const filter = event.target.elements.filter.value;

if (this.filter !== filter) {

this.filter = filter;

/\* notify the grid about the change \*/

this.params.filterChangedCallback();

}

}

}

我们现在已经完成了用户界面。

接下来，我们需要实现执行过滤的doesFilterPass函数。它由ag-Grid调用，以确定一个值是否通过当前筛选条件。我们需要实现的ag-Grid调用的另一个函数是isFilterActive。它确定筛选器是否具有要应用的筛选条件。

让我们实现这个功能在我们的组件:

export class RangeFilterComponent implements AfterViewInit {

filter = '';

params: any;

agInit(params: any): void {

this.params = params;

}

isFilterActive() {

return this.filter !== '';

}

doesFilterPass(params) {

const filter = this.filter.split('-');

const gt = Number(filter[0]);

const lt = Number(filter[1]);

const value = this.params.valueGetter(params.node); //单元格的当前值

return value >= gt && value <= lt;

}

...

}

获得单元格的当前值

注意，我们还通过参数从ag-Grid接收valueGetter函数。在下面的代码中，我们使用它来检索单元格的当前值。

过滤器预置aggrid实现的API，可用于激活和停用过滤器的需求。

通常，这个API是由一些UI元素触发的，比如一个按钮: 要使此功能工作，我们的自定义组件应该实现两个方法—setModel和getModel。

获得单元格当前过滤条件

getModel - 返回活动筛选器模型

ag-Grid调用它们来为组件提供当前过滤条件，或从组件获取该条件。

下面是我们如何在代码中实现这些方法:

export class RangeFilterComponent {

filter = '';

...

getModel() {

return {filter: this.filter};

}

setModel(model) {

this.filter = model ? model.filter : '';

}

}

我们需要做的最后一件事是把焦点放在文本框上。为此，我们将使用熟悉的ngAfterViewInit生命周期钩子:

@Component({

selector: 'app-range-filter-cell',

template: `

<form (submit)="onSubmit($event)">

<input #i name="filter" [value]="filter"/>

<button>Apply</button>

</form>

`

})

export class RangeFilterComponent implements AfterViewInit {

@ViewChild('i') textInput;

...

ngAfterViewInit() {

setTimeout(() => {

this.textInput.nativeElement.focus();

});

}

}

完整实现  
这是我们组件的完整代码:

@Component({

selector: 'app-range-filter-cell',

template: `

<form (submit)="onSubmit($event)">

<input #i name="filter" [value]="filter"/>

<button>Apply</button>

</form>

`

})

export class RangeFilterComponent implements AfterViewInit {

@ViewChild('i') textInput;

filter = '';

params: any;

agInit(params: any): void {

this.params = params;

}

isFilterActive() {

return this.filter !== '';

}

ngAfterViewInit() {

setTimeout(() => {

this.textInput.nativeElement.focus();

});

}

doesFilterPass(params) {

const filter = this.filter.split('-');

const gt = Number(filter[0]);

const lt = Number(filter[1]);

const value = this.params.valueGetter(params.node);

return value >= gt && value <= lt;

}

getModel() {

return {filter: this.filter};

}

setModel(model) {

this.filter = model ? model.filter : '';

}

onSubmit(event) {

event.preventDefault();

const filter = event.target.elements.filter.value;

if (this.filter !== filter) {

this.filter = filter;

this.params.filterChangedCallback();

}

}

}

注册组件  
一旦我们准备好了我们的组件，我们需要在frameworkComponents中注册它，并指定它为“Price”列的编辑器:

export class AppComponent implements OnInit {

columnDefs = [

{headerName: 'Make', field: 'make'},

{headerName: 'Model', field: 'model'},

{

headerName: 'Price',

field: 'price',

editable: true,

cellRenderer: 'numberFormatterComponent',

cellEditor: 'numericEditorComponent',

/\* custom filter \*/

filter: 'rangeFilterComponent'

}

];

frameworkComponents = {

/\* custom filtering component \*/

rangeFilterComponent: RangeFilterComponent,

};

用ag-Grid构建你自己的Angular应用!  
我希望上面的例子能清楚地说明定制和配置Angular网格是多么容易。