COVID

Github https://github.com/SFBB/COVID-Data-Visualization

目录

系统需求[目录]

- **对于疫情的蔓延进行时间上与地理上的描述**

  我们希望提供给用户可以在地理上理解疫情的严重以及不同国家地区疫情情况的不同。
  同时，我们也希望能够向用户展现在特定时间段上疫情蔓延程度的变化情况。

- **提供尽可能多工具来便于用户多方位对于上一点的理解**

  我们希望在一个简单的界面提供用户尽可能多的可视化工具满足用户尽可能多的需求。
  例如：

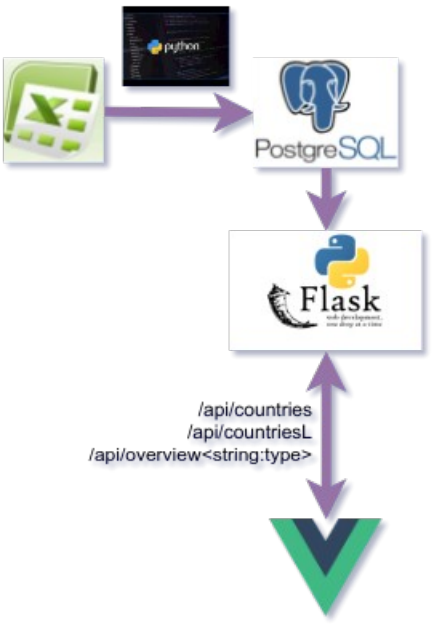
  - ✔ 展示给用户疫情整体情况的工具
  - ✔ 为用户提供展现特定时间段疫情的工具
  - ✔ 只展现特定范围的国家地区疫情的工具
  - ✔ 展现疫情严重国家地区的排序
  - ✔ 展现特定国家地区疫情的细节
  - ✔ ……

- **Out of Box 的架构，以利于之后的功能拓展和开发**

  我们希望这个系统是可以实际运行的，可以投入实际使用的。

  这是一个完整的可视化项目，可以很容易的扩展功能，例如：添加新的疫情数据，实现新的功能等。

系统框架[目录]



我们使用一个 python 脚本讲课程提供的数据进行提取并存储到 Postgres 数据库中。

我们使用 Flask 作为后端，与 Postgres 进行交流读取数据。

我们使用 VueJS 进行前端的渲染生成，通过后端提供的三个 API 接口与后端进行数据的申请传输。我们使用 AmCharts 进行可视化的绘制。

后端 —— 数据处理[目录]

1. 数据收集

使用的数据为课程网站上提供的 XLXS 存储的数据集。

使用一下方式将其转化为数据库标准格式进行存储，使用的数据库为 Postgres。[1]

- 人工过滤掉无用的 sheet，重复的 sheet。
- 使用 Python 的开源 Library --- Openpyxl 将数据从 xlxs 文件中进行提取，期间对只有日期的数据进行过滤。
- 最后通过 psycopg2 讲数据插入数据库。

本系统由于使用系统性架构，用户可自行向数据库中插入更多的数据，也可以自行编写相应的程序页面来便捷数据更新的过程，以实时追踪疫情的最新进展。

2. 数据库的应用

数据库拥有着 JSON 文件以及更加简陋的 XLXS 文件所不具备的优势，对于本系统而言，它体现在：

- 可以快速高效处理大量数据，*目前数据库中共存储有 20,863 行疫情数据。*
- 可以使用 SQL 语句轻松的实现对于数据的处理，本系统需要地理与时间两个维度的不同数据表示方式，同时需要将中文表示的国家地区与国家代码匹配，依据时间进行排序，数据也需要根据前端需求进行过滤与计算，如果通过 Python 或者其它预言进行实现，是一个不小的工作量，但是通过数据库 SQL 语句，一行就可以解决。
- 可拓展性。对于记录疫情数据，明显地我们希望可以对数据进行更新和完善，那么对于系统性保存数据并使以后可以对这些数据可以轻易更新便是需要的，数据库可以完好的满足这个需求

1 相关的代码位于 database/xlxs_2_db.py 中。

数据在数据库中的表示方法：

数据库名字:	yiqing	
table:	yiqing	
table yiqing:	id	对“(str(国家地区)+str(日期)).encode(“utf-8”)”进行 md5 哈希编码，确保每一个国家地区在同一个日期只有一条数据
	国家地区	中文表示的国家地区名字，例如美国、全球、巴西等。
	日期	以“yyyy-mm-dd”格式表示的日期数据
	累计确诊	int
	新增确诊	int
	累计死亡	int
	新增死亡	int
	累计治愈	int
	仍在治疗	int
	重症病例	int
	百万人口确诊率	float
	百万人口死亡率	float
	总检测数	float
	百万人口检测率	float

与数据库进行交互时可以参照上述内容进行处理。

国家地区代码的对应

对于国家的代码我们使用位于 back/a.json 中的*国家地区-国家代码*对照表来对应。相关代码在 back/run.py 中。

```
def get_short(data):
    for cou in ZH_2_EN:
        if cou["name"] == data[0]:
            return cou["short"]
    return "None"

def get_name(short):
    for cou in ZH_2_EN:
        if cou["short"] == short:
```

```
        return cou["name"]
    return None
```

如果想要使用本项目的地图模块，这一步是必需的，而且需要随着数据的更新不断更新这个对照表，因为绘制地图时便是根据国家代码作为索引绘制的，这一点会在前端部分进行介绍。

3. API 接口

目前后端共提供三个 API 接口来满足目前的系统需求。需要扩展时可以自行添加，这是很容易的步骤。

API 类型	API 作用	参数要求	返回数据格式
/api/countries_	以国家为单位，返回对应时间段的这些国家总的新增确诊，新增死亡，重症病例，累计确诊，累计治愈，仍在治疗，累计死亡。	?from=要求时间段的起始时间(*yyyy-mm-dd*)&to=要求时间段的结束时间(*yyyy-mm-dd*)，无参数(*默认处理为目前数据库中的所有时间*)	{countries: [{id[2], name[3], 仍在治疗, 新增死亡, 新增确诊, 累计死亡, 累计治愈, 累计确诊, 重症病例}, ...]}
/api/overview_	以时间为单位，返回所有国家地区(除全球)在这些时间上的请求的平均新增确诊，新增死亡，重症病例，累计确诊，累计治愈，仍在治疗，累计死亡。	type(*需要显示的疫情项目*)	{dates: [{date[4], 仍在治疗, 新增死亡, 新增确诊, 累计死亡, 累计治愈, 累计确诊, 重症病例 }, ...]}
/api/countriesL	以时间为单位，返回请求的国家列表对应的新增确诊，新增死亡，重症病例，累计确诊，累计治愈，仍在治疗，累计死亡。	?from=要求时间段的起始时间(*yyyy-mm-dd*)&to=要求时间段的结束时间(*yyyy-mm-dd*)&countries=countA,countB,countC[5]...(*请求的国家列表，后端只会返回国家列表中的国家的数据*)&type=疫情的项目(*例如：新增确诊、累计死亡等*)	{countries: {countA: [{date, 仍在治疗, 新增死亡, 新增确诊, 累计死亡, 累计治愈, 累计确诊, 重症病例}, ...], countA: [...], countA: [...], ...}}

2 为国家代码，例如 CN，JP，US 等。

3 为国家地区名称，例如中国，日本，美国等。

4 为表示日期的信息，使用 Python3 中 int(time.mktime(date[7].timetuple()))*1000 进行转换以方便前端解析。

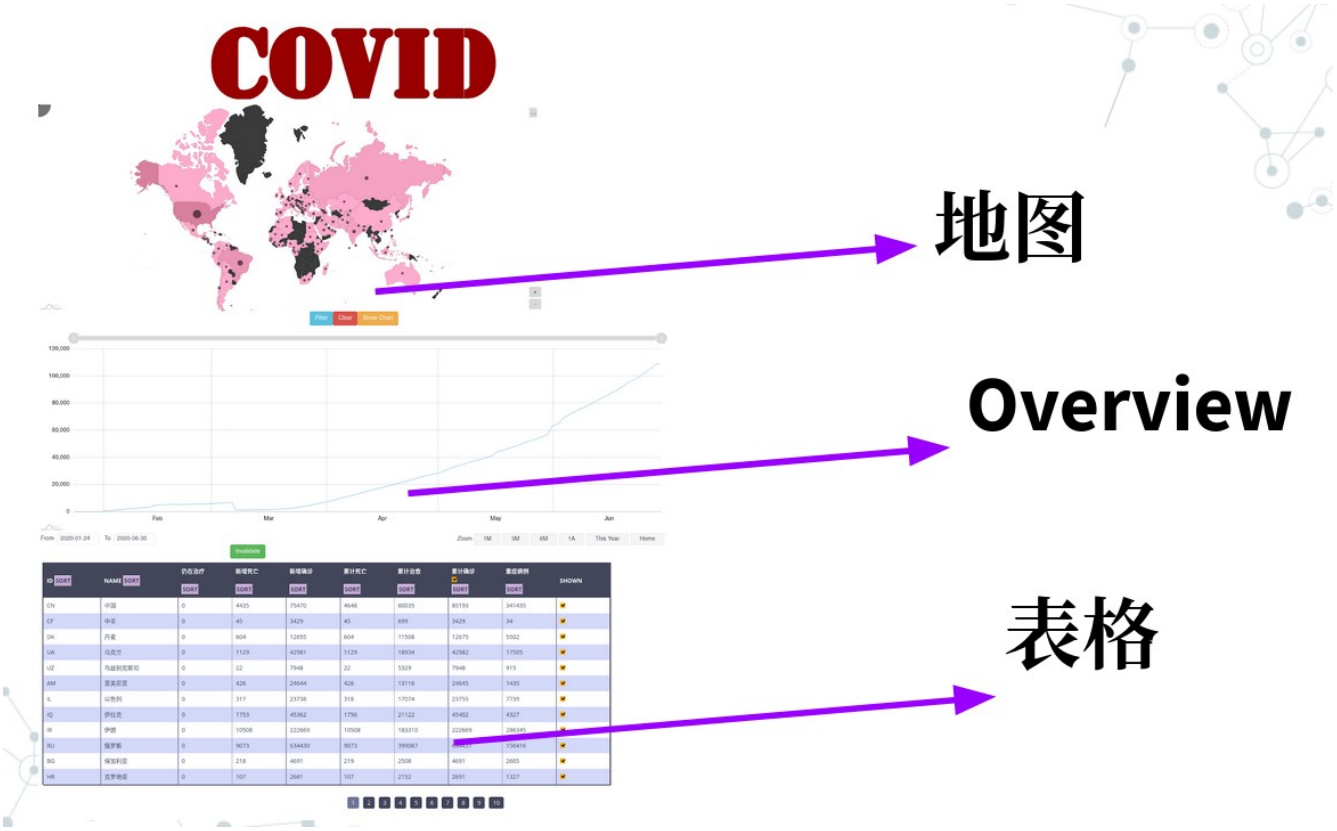
5 countA, countB, countC, … 使用国家代码表示，例如: CN, JP, US, …。

前端可以通过调用这些 API 实现与后端的通信，后端受到请求后会根据 API 的类别与 PostgresDB 进行通信，使用对应的 SQL 语句索取相应的数据，并经过初步处理返回给前端。相关代码在 back/run.py 中。

具体访问方法可以见前端部分的介绍。

前端 —— 功能实现[目录]

1. 总览



前端共分三个模块：

- 地图
- Overview
- 表格

我们可以将*地图*理解为**展示面板**，将 *Overview+表格*理解为**控制面板**，同时**展示面板**附带一些对于局部数据的控制展示功能。

我们将会基于这些面板来介绍前端的功能，这样可以更好的理解本系统的操作。

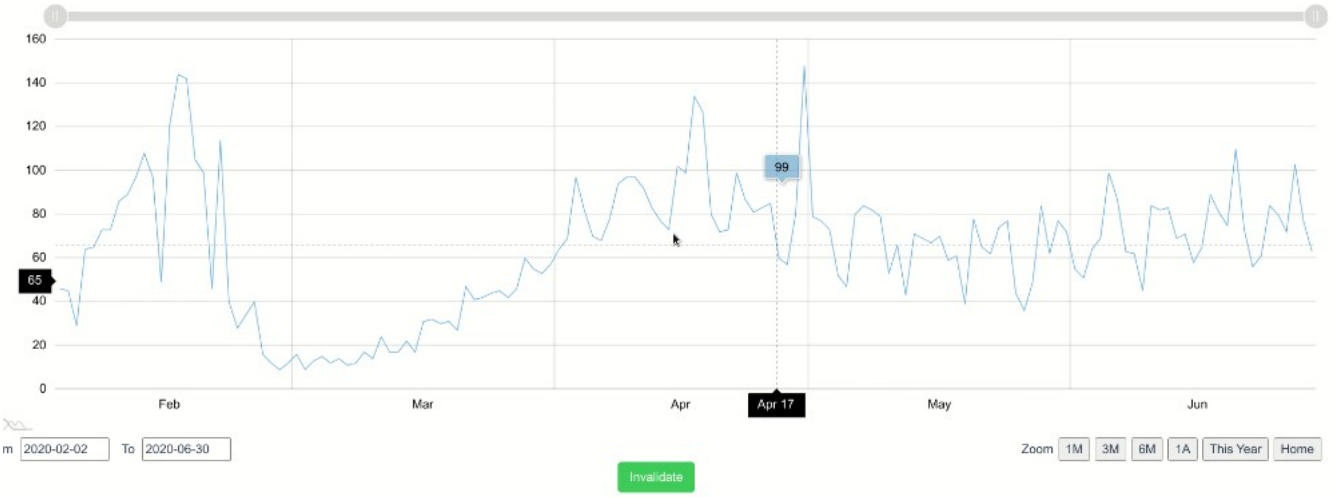
2. 控制面板

控制面板提供较少的可视化组件，但是并非没有。

控制面板的主要目的是给用户直观的对于目前系统所成呈现数据的理解和概览，提供用户多种对于目前所展示数据的操作选项。

控制面板由 **Overview** 和**表格**组成。

Overview



Overview 提供了对于目前数据的总体可视化概览。它会生成一张折线图，对于目前所展示的疫情项目[6]基于所选定的时间段，展示该疫情项目(*对全球除外的国家地区的各项项目取平均或者最大值*)随时间的变化情况。

同时 Overview 提供了对于全局数据时间段的控制，当用户通过滑动、输入方式制定一段时间段时，在点击 **Invalidate** 这个 Button 后，就可以更新全局数据，使这些数据为这些时间段的内容。

相关演示请见 *table-overview.mp4*。

实现方法

相关代码在 *front/src/components/CountryTable.vue* 中。

6 疫情项目指的是反应疫情的几个种类，例如：新增确诊，累计确诊，累计死亡等，这些信息在表格的头进行展现，并且表格为用户提供了复选框，以供用户选择目前所要展示的疫情项目。详见**表格**相关的介绍。

初始化 Overview：

```
axios.get("http://127.0.0.1:5000/api/overview"+this.showing) // this.showing记录了目前选中的疫情项目
.then( function(Response) {
am4core.useTheme(am4themes_animated); // 选择相应的主题

var chart = am4core.create("couOver", am4charts.XYChart); // 在id为couOver的元素下创建XY Chart
this.chart = chart; // 使用全局变量记录这个chart，方便以后的数据更新
// console.log(chart);

// chart.language.locale = am4lang_es_ES;

chart.paddingRight = 20; // 右padding的设置

// 无用
// var data = [];
// var visits = 10;
// for (var i = 1; i < 50000; i++) {
// visits += Math.round((Math.random() < 0.5 ? 1 : -1) * Math.random() * 10);
// data.push({ date: new Date(2018, 0, i), value: visits });
// }

var dates_t = [...Response.data.dates] // 复制返回的数据，安全方式，防止原始数据被影响
dates_t.forEach(function(date, index){
Response.data.dates[index].date = new Date(date.date); // 解析返回的日期，使其符合js格式
});
// console.log(data);
chart.data = Response.data.dates; // 指定chart的data
// console.log(Response.data.dates);

var dateAxis = chart.xAxes.push(new am4charts.DateAxis()); // 绘制Date轴
dateAxis.renderer.grid.template.location = 0; // 制定位置
dateAxis.minZoomCount = 5; // 指定最小缩放的距离


// this makes the data to be grouped
dateAxis.groupData = true;
dateAxis.groupCount = 500;

var valueAxis = chart.yAxes.push(new am4charts.ValueAxis()); // 添加数值轴
```

```
var series = chart.series.push(new am4charts.LineSeries()); // 添加 Line 序列
this.series = series; // 使用全局变量保存，便于以后更新
series.dataFields.dateX = "date"; // 制定 X 轴为数据中的 date 项
series.dataFields.valueY = this.showing; // 制定 Y 轴为目前显示的疫情项目
series.tooltipText = "{valueY}";
series.tooltip.pointerOrientation = "vertical";
series.tooltip.background.fillOpacity = 0.5;

chart.cursor = new am4charts.XYCursor();
chart.cursor.xAxis = dateAxis;

var scrollbarX = new am4core.Scrollbar();
scrollbarX.marginBottom = 20;
chart.scrollbarX = scrollbarX;

// 添加 range selector
var selector = new am4plugins_rangeSelector.DateAxisRangeSelector();
this.selector = selector;
selector.container = document.getElementById("selectordiv"); // 位置设定在这个元
素下
selector.axis = dateAxis;

// 添加几个便捷按钮
chart.language.setTranslationAny("%1Y", "%1A");
chart.language.setTranslationAny("%1M", "%1M");
chart.language.setTranslationAny("YTD", "This Year");
chart.language.setTranslationAny("MAX", "Home");
}.bind(this)); // 将 this 传入函数
```

Invalidate 按钮使用 refresh 函数更新全局数据:

```
"refresh": function refresh() {
// 获取目前指定的时间段的起始日期
var from = document.getElementsByClassName("amcharts-range-selector-from-
input")[0].value;
// 获取目前指定的时间段的终止日期
var to = document.getElementsByClassName("amcharts-range-selector-to-input")
[0].value;
// console.log(from+to);
// 利用 API 请求后端数据
axios.get("http://127.0.0.1:5000/api/countries_?from="+from+"&to="+to)
.then( function(Response) {
this.rows = Response.data.countries; // 更新 Table 中的数据
this.realrows = [...this.rows]; // 对于目前 Table 数据的备份，在其它的功能会用到
```

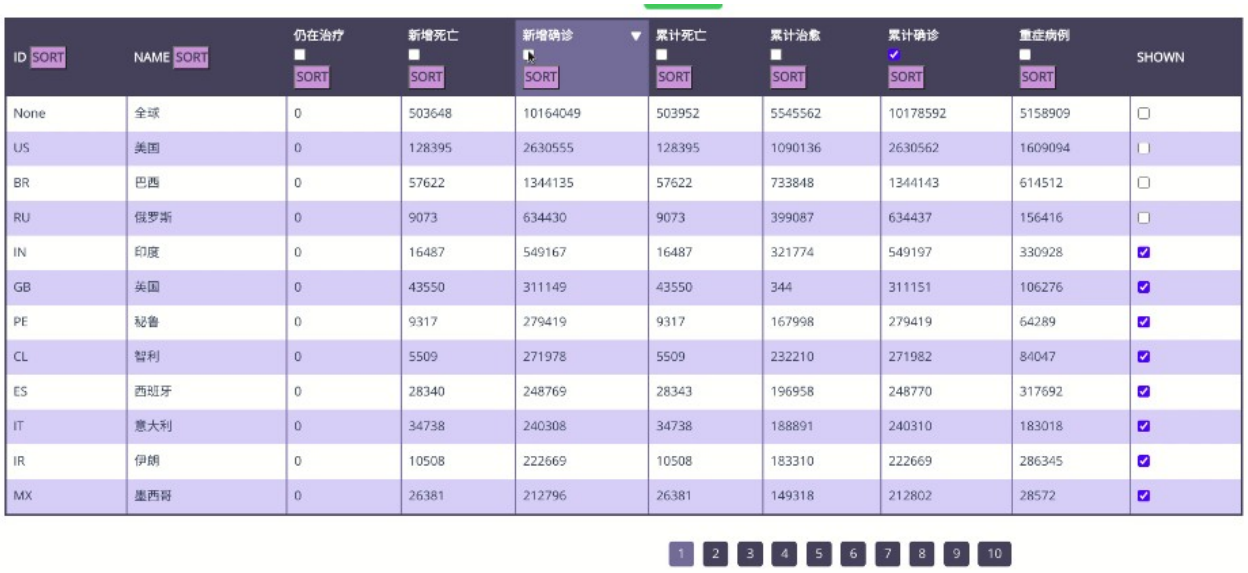
```
var temp = [...this.rows]; // 更新后的数据
// var temp = [this.rows]
// console.log(Response.data.countries);
this.$root.$emit('date_updated', temp, this.Shown); // this.Shown 为 Table 中用户想要显示的国家地区。
HelloWorld.vue，更新地图数据。
}.bind(this));
// console.log("Refresh!");
},
```

下面的是在 HelloWorld.vue 中的相关代码:

```
this.$root.$on('date_updated', function(new_data, Shown) {
console.log("sadasd!!!");
// console.log(new_data);
this.realdata = [...new_data]; // 同样对于数据的备份，其它功能会用到
// polygonSeries.data = JSON.parse(JSON.stringify(new_data));
// bubbleSeries.data = JSON.parse(JSON.stringify(new_data));
this.MapData = new_data; // 更新地图数据
console.log(new_data);
(async function() {
console.log(this.MapData);
const dogs = await reDraw(Shown, this.MapData); // 根据需要显示的国家地区和地图数据对地图重新绘制
// console.log(dogs)
}.bind(this))()
}.bind(this));

async function redraw(Shown, data){
// console.log(Shown);
// console.log(Object.keys(Shown).length);
// console.log(data);
var temp = [...data];
// console.log(temp);
data.forEach(function(cou, ind){
// console.log(Shown[cou.id]);
// 根据是否显示，对数据进行过滤
if(!Shown[cou.id]){
// console.log(cou.id);
if(temp.indexOf(cou) > -1)
temp.splice(temp.indexOf(cou), 1);
}
});
// this.realdata = [...temp];
data = temp;
this.MapData = temp;
```

```
// console.log(data);
polygonSeries.data = JSON.parse(JSON.stringify(this.MapData)); // 更新多边形的数据，多国家地区颜色进行重新绘制
bubbleSeries.data = JSON.parse(JSON.stringify(this.MapData)); // 更新泡泡的数据，对于表示疫情的泡泡进行重新绘制
// func("Done!");
return "Done!";
}

let reDraw = redraw.bind(this); // 将 this 传入函数
```

表格



ID SORT	NAME SORT	仍在治疗 SORT	新增死亡 SORT	新增确诊 SORT	累计死亡 SORT	累计治愈 SORT	累计确诊 SORT	重症病例 SORT	SHOWN
None	全球	0	503648	10164049	503952	5545562	10178592	5158909	☐
US	美国	0	128395	2630555	128395	1090136	2630562	1609094	☐
BR	巴西	0	57622	1344135	57622	733848	1344143	614512	☐
RU	俄罗斯	0	9073	634430	9073	399087	634437	156416	☐
IN	印度	0	16487	549167	16487	321774	549197	330928	☑
GB	英国	0	43550	311149	43550	344	311151	106276	☑
PE	秘鲁	0	9317	279419	9317	167998	279419	64289	☑
CL	智利	0	5509	271978	5509	232210	271982	84047	☑
ES	西班牙	0	28340	248769	28343	196958	248770	317692	☑
IT	意大利	0	34738	240308	34738	188891	240310	183018	☑
IR	伊朗	0	10508	222669	10508	183310	222669	286345	☑
MX	墨西哥	0	26381	212796	26381	149318	212802	28572	☑

1 2 3 4 5 6 7 8 9 10

表格详列了目前显示的数据。

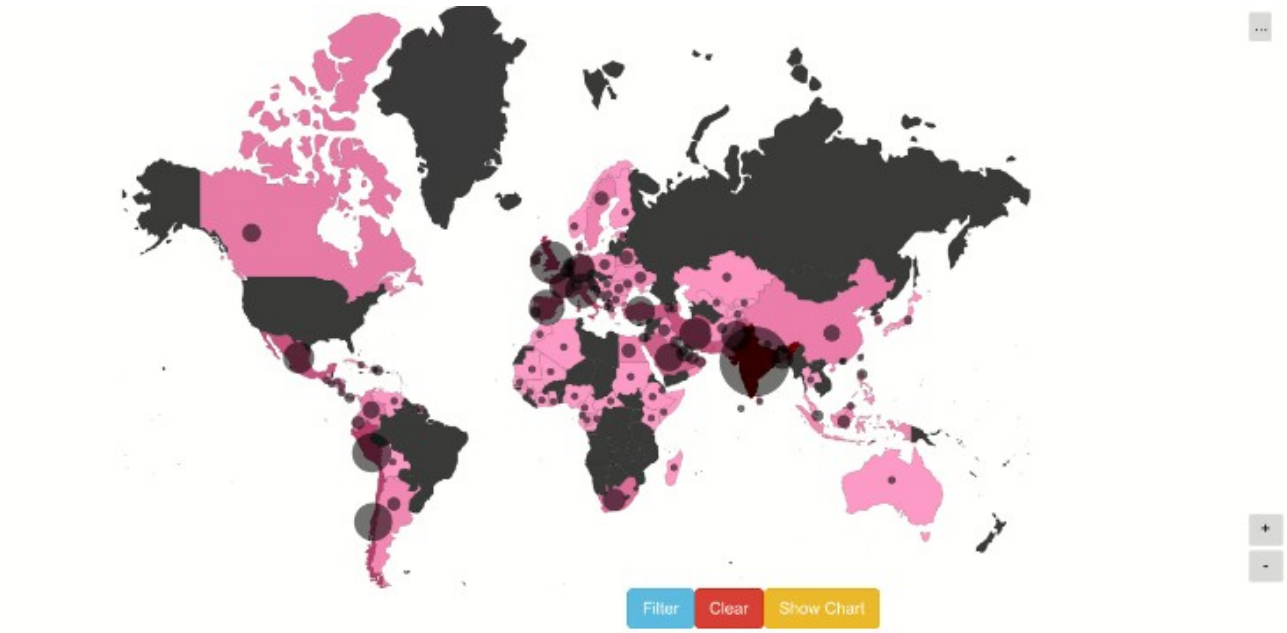
表格提供了表格的基础功能(排序，翻页，详见 *table-basic.mp4*)，对于要显示的国家地区的控制(*table-shown.mp4*)，对于目前显示的疫情项目的控制(*table-key.mp4*)。

表格会随着 **Overview** 的更新而更新，也会随着**地图** Filter 功能的使用而更新。

相关演示请见 *table-*.mp4*。

相关功能的具体实现

对于要显示的国家地区的控制



疫情严重的国家的数据会影响到其它国家疫情情况对比度的展示，不显示某些疫情严重国家的数据可以更加清晰的看到仍然很严重但是又不是最严重的国家的疫情对比情况。

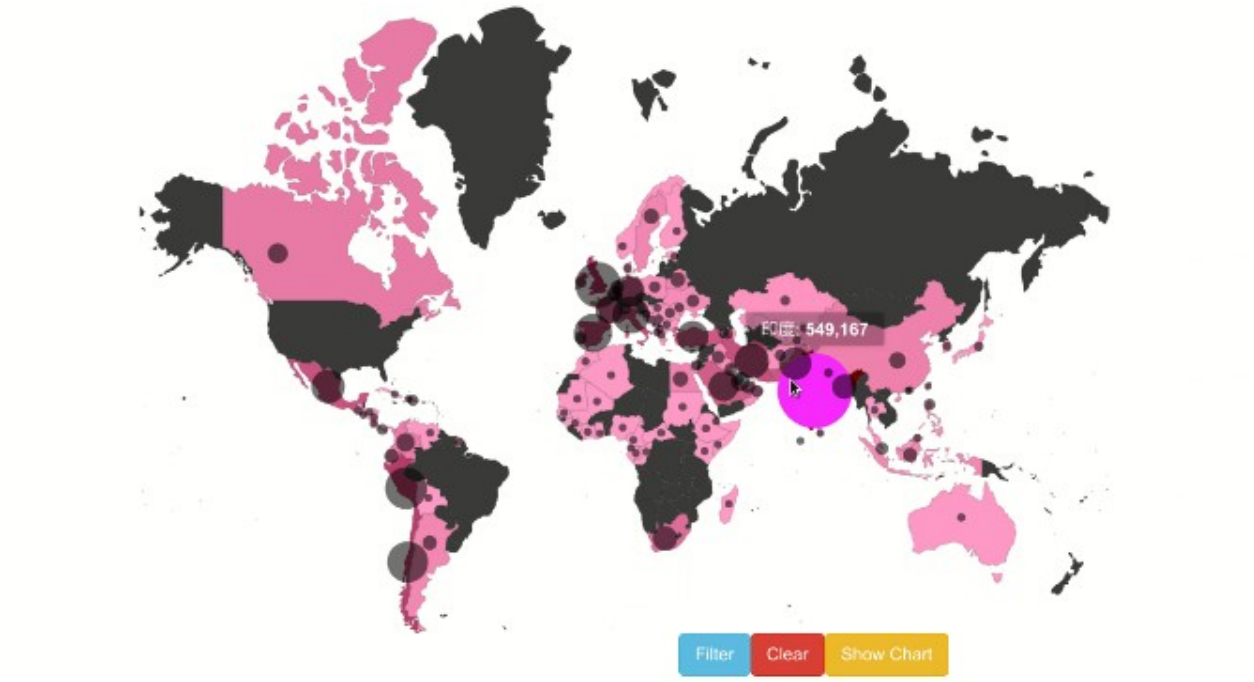
点击表格最后一列的 Shown 列的 checkbox 都会触发这个函数，通过 country_id 来确认是哪个国家地区，并更新数据。

```
"shown": function shown(country_id) {
this.Shown[country_id] = !this.Shown[country_id]; // 对于目前国家地区的显示与否更新
// console.log(this.Shown[country_id]);
// console.log(HelloWorld.computed.get_general());
this.$root.$emit('shown', this.Shown); // 发送一个信号给等待接收的地图那边，以更新地图数据
},
```

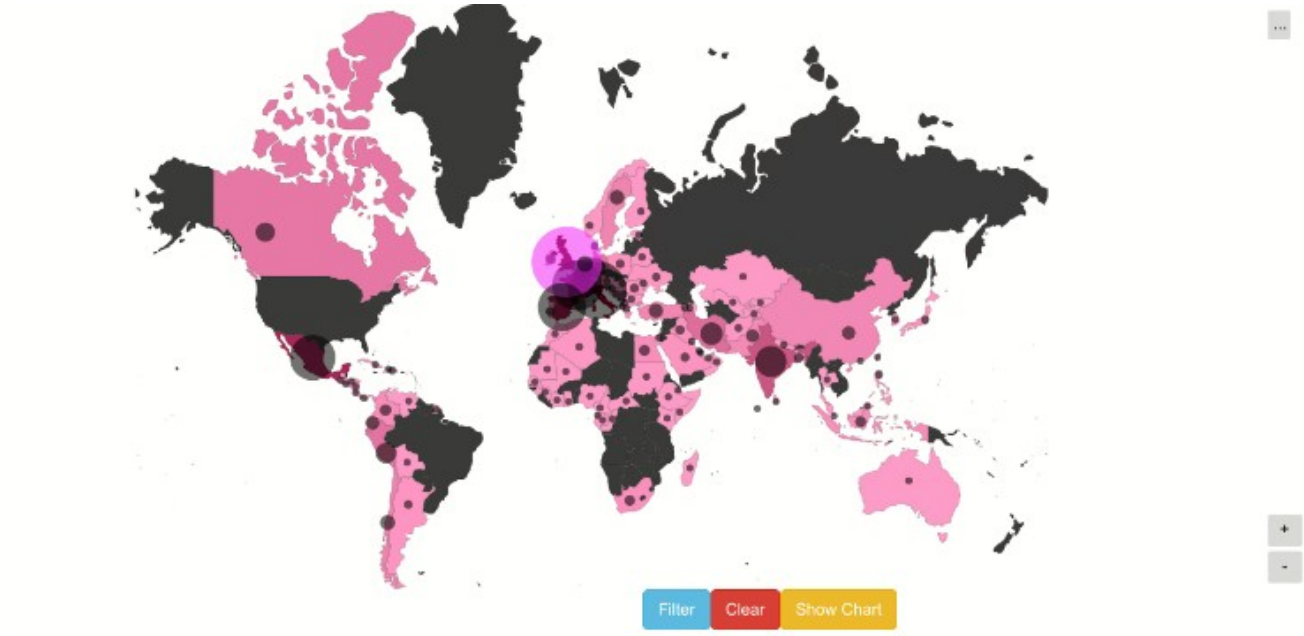
下面的是在 HelloWorld.vue 中的相关代码:

```
this.$root.$on('shown', function(Shown) {
// bubbleSeries.data = temp;
console.log(this.MapData.length);
(async function() {
const dogs = await reDraw(Shown, this.MapData); // 重新绘制地图
// console.log(dogs)
}.bind(this))()
console.log(this.MapData.length);
// console.log("Received!");
// console.log(temp);

}.bind(this));
```

对于目前显示的疫情项目的控制



新增确诊



新增死亡

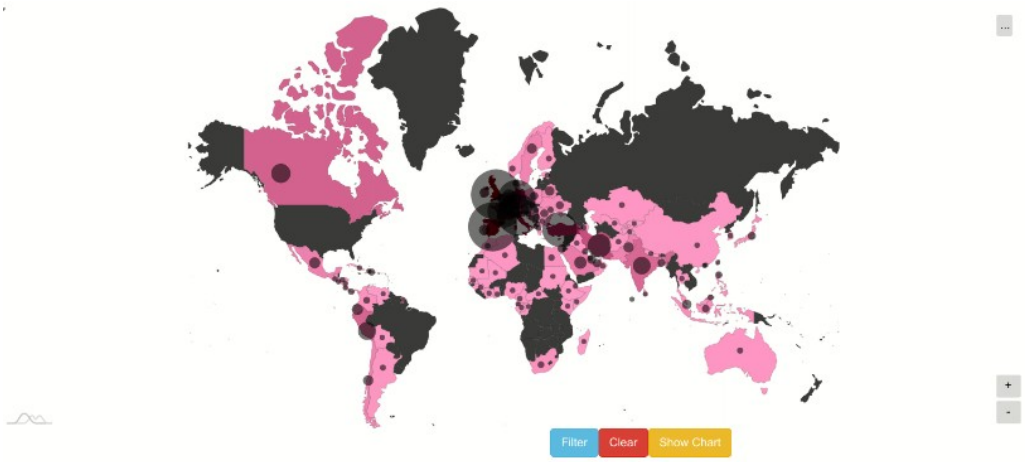
提供可交互式的界面以满足用户想要查看不同疫情项目的需求。

```
"datashown": function datashown(data_to_show) {
this.Data_Showing[data_to_show] = !this.Data_Showing[data_to_show]; // 将被点击的那一列对应的疫情项目取反
this.showing = data_to_show; // 更新目前正在显示的疫情项目
// this.Data_Showing["累计确诊"] = false;
// 更新其它疫情项目为 false，以保证只显示一项疫情项目
Object.keys(this.Data_Showing).forEach(function(key, index){
if(key != data_to_show){
this.Data_Showing[key] = false;
}
}.bind(this));
this.$root.$emit('datashowing', data_to_show); // 发送一个信号给等待接收的地图那边，以更新地图数据
this.update_overview(); // 更新 Overview
},

"update_overview": function update_overview() {
// 向后端请求目前显示的疫情项目的相关数据。并更新 Overview
axios.get("http://127.0.0.1:5000/api/overview"+this.showing)
.then( function(Response) {
this.chart.data = Response.data.dates;
this.series.dataFields.valueY = this.showing;
}.bind(this));
},
```

下面的是在 HelloWorld.vue 中的相关代码:

```
this.$root.$on('datashowing', function(data_to_show) {
// console.log(data_to_show);
this.showing = data_to_show; // 更新目前正在显示的疫情项目
bubbleSeries.dataFields.value = data_to_show; // 更新泡泡数据需要显示的疫情项目
polygonSeries.dataFields.value = data_to_show; // 更新多边形需要现实的疫情项目
// this.MapData = [...this.realdata];
// 更新数据，否则无法重绘
polygonSeries.data = JSON.parse(JSON.stringify(this.MapData));
bubbleSeries.data = JSON.parse(JSON.stringify(this.MapData));

}.bind(this));
```

3. 展示面板



展示面板只含有地图模块。

地图为用户提供了丰富的可视化信息。地同通过国家地区颜色的深浅不同表示不同国家在相应的时间段、目前正在显示的疫情项目的对比，通过泡泡展现了疫情的严重程度以及不同国家地区间的对比。

用户可以通过悬浮鼠标到特定位置了解某个国家疫情情况的细节，地图也支持缩放功能。(*map-basic.mp4*)

同时用户可以选择想要详细了解的国家地区，通过 Show Chart 按钮，展现这几个国家地区在目前的时间段所有疫情项目随着时间变化的折线图(*第一个表格为目前正在显示的疫情项目*)，这些表格功能全面，支持是否绘制某些国家地区，随着鼠标移动而现实数据细节信息，以及缩放等功能。(*map-chart.mp4*)

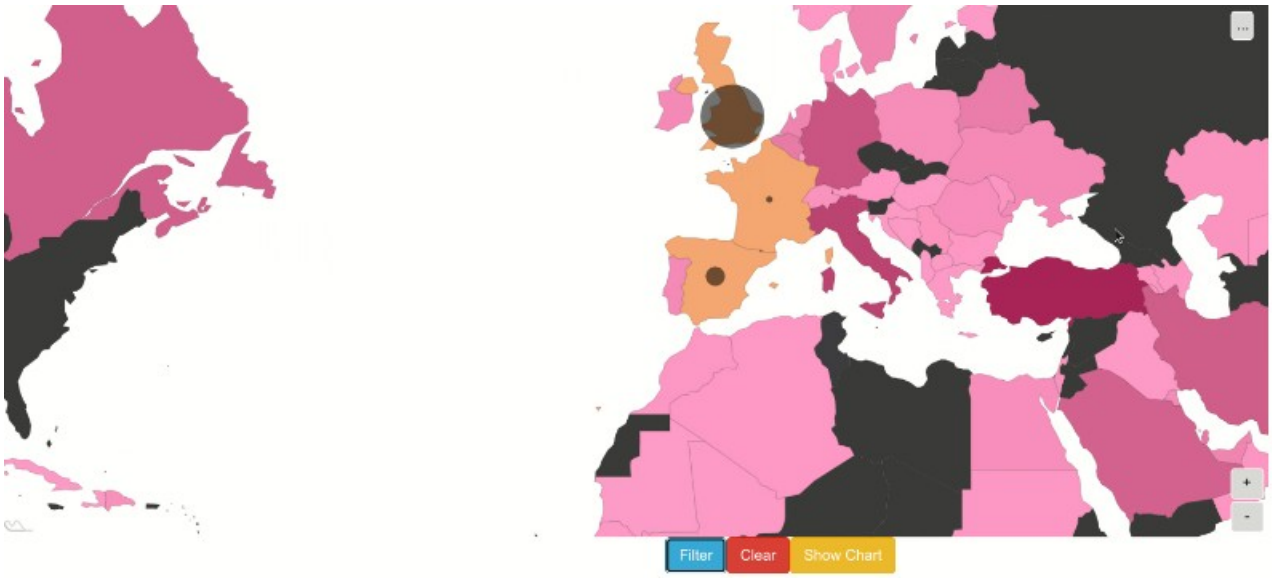
地图还为用户提供 Filter 功能，当用户想要只比较部分国家地区(*过多的国家地区同时显示会使用户难以看出疫情较不为严重的国家地区们的疫情对比情况*)，可以通过这个功能实现，放心，这不会影响全局数据。(*map-filter.mp4*)

相关演示请见 *map-*.mp4*。

相关功能的具体实现

相关代码在 *front/src/components/HelloWorld.vue* 中。

Map Filter



ID SORT	NAME SORT	仍在治疗 SORT	新增死亡 SORT	新增确诊 SORT	累计死亡 SORT	累计治愈 SORT	累计确诊 SORT	重症病例 SORT	SHOWN
UA	乌克兰	0	541	18387	579	6227	19706	5680	☑
AT	奥地利	0	413	4107	633	14951	16404	6358	☑
DE	德国	0	6574	76568	8309	158000	179021	120450	☑
PL	波兰	0	865	15730	972	8452	20143	7090	☑
BY	白俄罗斯	0	172	32671	185	12057	33371	3697	☑

通过 Filter 功能，用户可以查看少数几个国家的疫情情况，对比关系在同中更加明确，同时表格也被过滤，只剩下用户选择的国家，可以是用户更快速的浏览这些国家在这个时间段的大体情况。

通过对每一个多边形添加 hit event，来实现对于国家地区的选择功能。

```
polygonSeries.mapPolygons.template.events.on("hit", function(ev) {
console.log(ev.target.isActive);
this.Shown[ev.target.dataItem.dataContext.id] = !ev.target.isActive; // 记录这个国家地区是否要显示
this.records[ev.target.dataItem.dataContext.id] = ev.target; // 记录被点中的国家地区
}.bind(this));
```

```
"dataShown": function dataShown(condition) {

var from = document.getElementsByClassName("amcharts-range-selector-from-
```

```
input")[0].value;
var to = document.getElementsByClassName("amcharts-range-selector-to-input")
[0].value;

var temp = [...this.MapData];
this.MapData.forEach(function(cou, ind){
if(!this.Shown[cou.id] && condition){ // 根据 Shown 和 condition 来过滤数
据，condition 为 true 的情况，只需根据 Shown 的情况来显示要显示的少数国家地区，如果为
false，那么就是 clear 功能，无视掉 Shown 的内容，显示所有国家地区(表格允许显示的)

if(temp.indexOf(cou) > -1)
temp.splice(temp.indexOf(cou), 1);
}
if(this.records[cou.id])
this.records[cou.id].isActive = this.Shown[cou.id]; // 是否显示选中
}.bind(this));

this.bubbleSeriess.data = JSON.parse(JSON.stringify(temp)); // 只更新泡泡的数据，
因为是局部功能，方便用户仍然能够通过来选择下一步操作
this.$root.$emit('filter', this.Shown); // 发送一个信号给正在接收信号的 Table 那边，
已更新 Table 的数据为要显示的少数国家的详细信息
},
```

CountryTable.vue 中的接收 filter 信号的相关函数。

```
this.$root.$on('filter', function(Shown) {
this.filter(Shown);
}.bind(this));

"filter": function filter(Shown) {
var temp = [...this.realrows]; // 复制，以免影响原始数据，同时方便之后复原/clear
this.realrows.forEach(function(cou, ind){

if(!Shown[cou.id]){

if(temp.indexOf(cou) > -1)
temp.splice(temp.indexOf(cou), 1);
}

}.bind(this));
this.rows = temp;
},
```

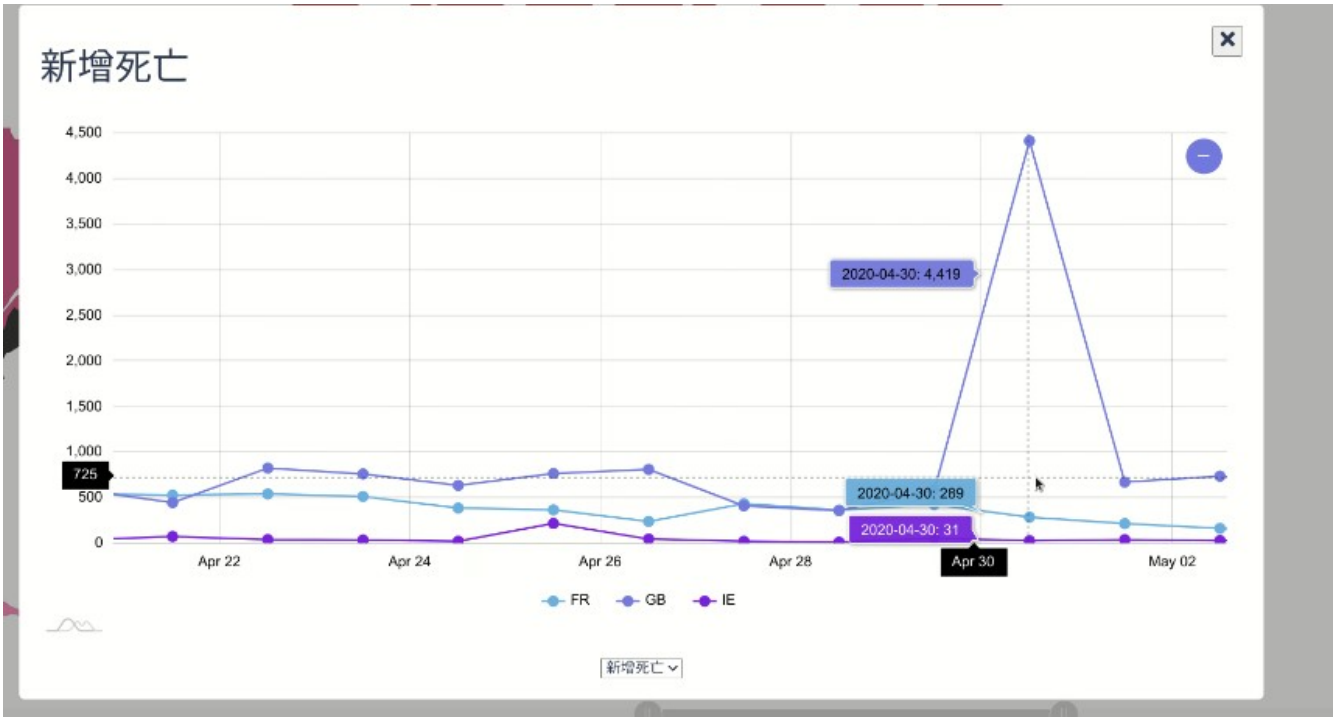
实现对于过滤器的清除/Clear 功能

```
"clear": function clear() {
this.realdata.forEach(function(cou){

this.Shown[cou.id] = false; // 全部不选中
}.bind(this));
this.dataShown(false); // 重新调用 dataShown 函数，这次 condition 为 false，相关内容在这个函数中已经解释了
this.$root.$emit('clear'); // 发送 clear 信号给 table
},
```

CountryTable.vue 中的接收 clear 信号的相关函数。

```
this.$root.$on('clear', function() {
this.clear();
}.bind(this));


"clear": function clear() {
this.rows = [...this.realrows]; // 恢复备份的原始数据即可
}
```

Map Chart



用户可以在图中下面的选项框中选择要看的数据。

这个功能为用户提供选中的国家各项疫情项目随着时间的变化情况。

按下 Show Chart 按钮后调用下面的函数初始化表格，第一张表格内容为目前正在显示的疫情项目。

```
"showchart": function showchart() {
var countries = {};
// 获取 from 和 to 的值
var from = document.getElementsByClassName("amcharts-range-selector-from-
input")[0].value;
var to = document.getElementsByClassName("amcharts-range-selector-to-input")
[0].value;
// 得到要获取数据的国家列表
Object.keys(this.Shown).forEach(function(cou){
if(this.Shown[cou]){
countries[cou] = [];
}
}.bind(this));
var couL = "";
Object.keys(countries).forEach(function(cou){
couL += cou+",";
});
couL = couL.slice(0, couL.length-1);
var showingg = this.showing;
// 如果确实选中国家的话
if(couL!="")
{
this.show = true; // 展示相应的 Modal
// 向后端请求数据
axios.get("http://127.0.0.1:5000/api/countriesL?
from="+from+"&to="+to+"&countries="+couL+"&type="+this.showing)
.then( function(Response) {

// 对数据进行处理以填入 XY Chart
var data = [];
Response.data.countries[Object.keys(Response.data.countries)
[0]].forEach(function(cou, index){
var temp = {};
temp.date = cou.date;
temp[Object.keys(Response.data.countries)[0]] = cou[showingg];
data.push(temp)
});
```

```
for(var i=1; i<Object.keys(Response.data.countries).length; i++){
var values = Response.data.countries[Object.keys(Response.data.countries)[i]];
var indexes = values.map(obj => obj.date);
data = data.map(obj => {
var index = indexes.indexOf(obj.date);
var cou = Object.keys(Response.data.countries)[i];
var temp = {};
temp[Object.keys(Response.data.countries)[i]] = index > -1? values[index]
[showingg] : obj[showingg];
return Object.assign({}, obj, temp);
});
}




var chart = am4core.create("DetailsChart", am4charts.XYChart);

// 添加 data
chart.data = data;

// 创建轴
var categoryAxis = chart.xAxes.push(new am4charts.DateAxis());
categoryAxis.renderer.grid.template.location = 0;

var valueAxis = chart.yAxes.push(new am4charts.ValueAxis());

// 创建序列函数
function createSeries(field, name) {
var series = chart.series.push(new am4charts.LineSeries());
series.dataFields.valueY = field;
series.dataFields.dateX = "date"; // X轴为日期
series.name = name;
series.tooltipText = "{dateX}: [b]{valueY}[/]";
series.strokeWidth = 2;
var bullet = series.bullets.push(new am4charts.CircleBullet());
bullet.events.on("hit", function(ev) {
alert("Clicked on " + ev.target.dataItem.dateX + ": " +
ev.target.dataItem.valueY);
});
}

var keyss = Object.keys(data[0]);
for(i=1; i<keyss.length; i++){
createSeries(keyss[i], keyss[i]);
```

```
}

chart.legend = new am4charts.Legend();
chart.cursor = new am4charts.XYCursor();
});
}
else
alert("Please select at least one country!");
},
```

当用户切换表格时会通过 changeChart 函数完成，内容几乎遇上面的函数一致，只是关于请求数据的种类上，是通过获取 select 元素的值实现的，这里不再详述。