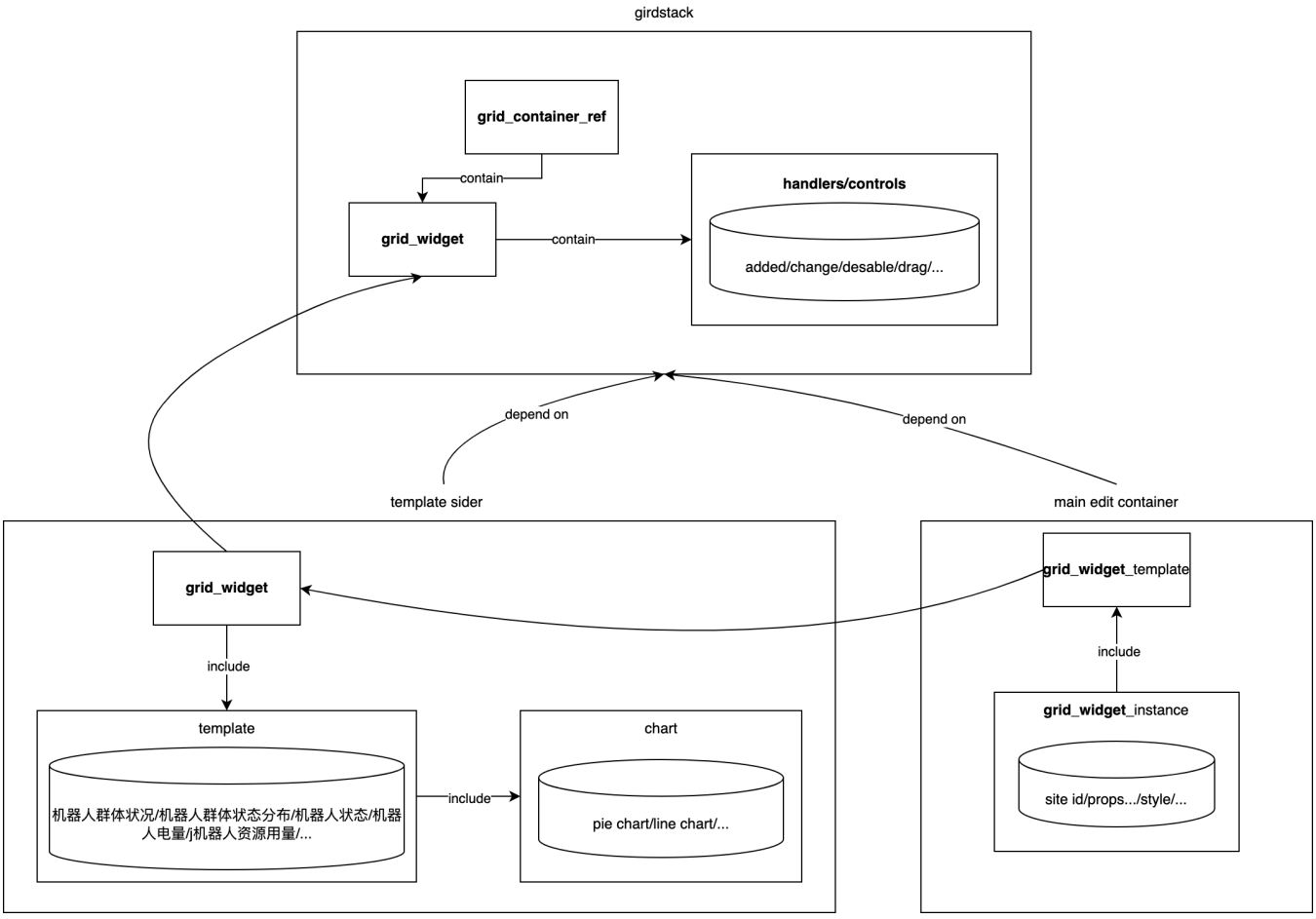
dataCenter前端技术详细设计

整体架构

UI 层架构



gridstack

gridstack：可以将 gridstack 理解成一个提供可编辑 widget 的画布，他必须依赖于一个叫 grid_container 的元素。

grid_widget: 可以将这个理解为在 grid_container 画布下的元素，这些元素都是可以被编辑的，这些元素可以被增加了一些事件，同时我们监听到这些事件。

这些事件包括：

1. 拖拽
2. 禁用
3. 增加
4. 改变
5. ...

template sider

sider：这里可以理解成左侧的模板列表 ，这里放的都是一些模板，但是由于这些模板也可以被拖动，所以他们同样继承与上面的基础 gridstack 类中的 grid_widget。可以简单的理解成 template sider widget with grid_widget。

template sider widget：中可以包含任何东西，首先他包含了一个模板，该模板告诉我这是一个关于机器人群体状况的模板，他有自己的 **模板 id，**通过模板 id 我们可以知道他是什么模板。

chart：在 template sider widget 中包含了他可以有的 chart 比如在 机器人群体状况的这个模板中可以包含一个饼图或者折线图等模板本身可以不关心他们相互都是独立又是可以相互组合的。

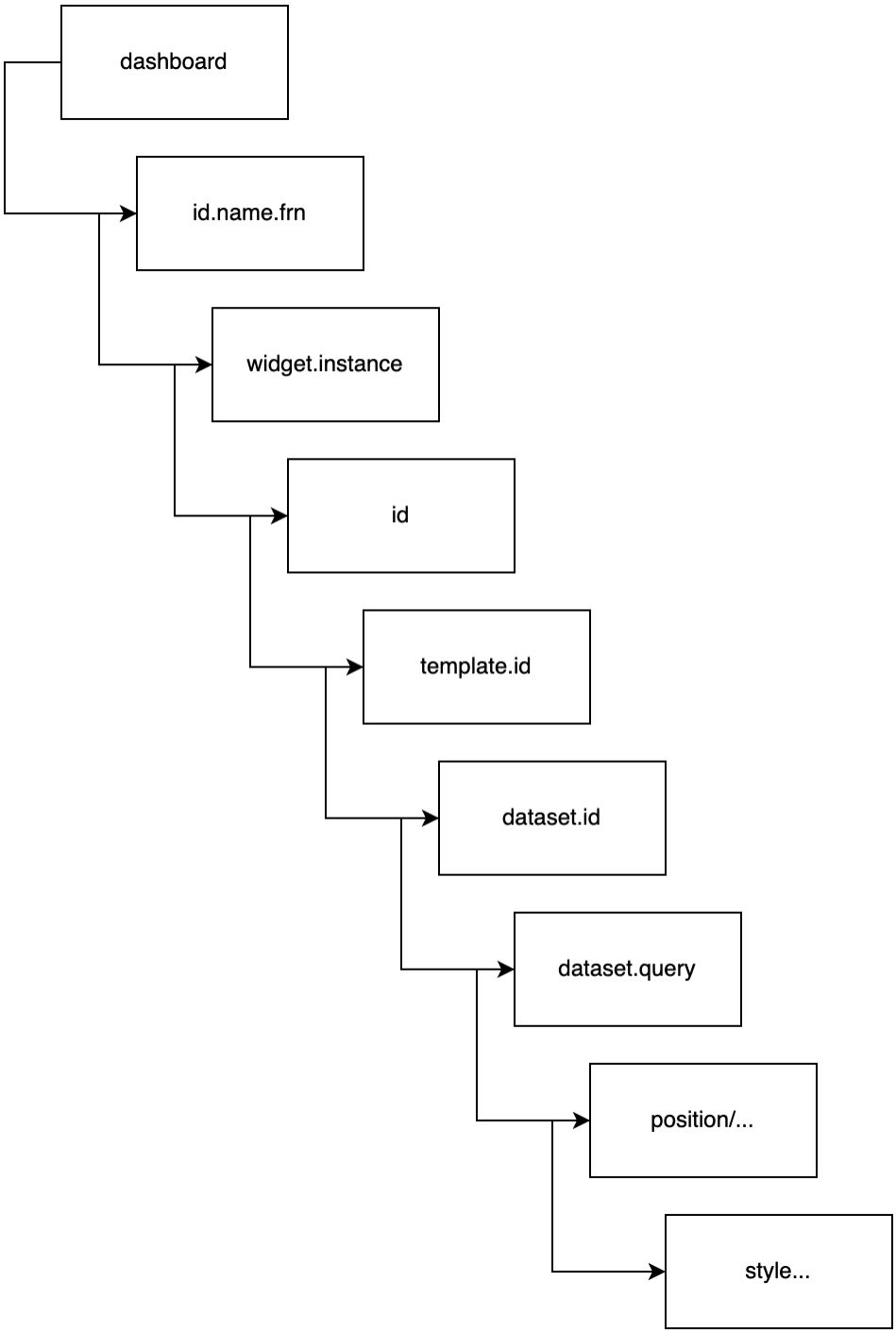
grid_widget中可以没有template，template中也可以没有 chart，他们都独立负责自己应该处理的逻辑，同时需要彼此分开，这就是程序上的正交。

main edit container

grid_widget_template：这些 widget_template 在没有指定如何获取其真正的数据时都是 template。

grid_widget_instance：一旦template 指定了数据的来源后他就是实例，实例里面有他应有的模板，以便可以重新指定数据来源。

数据层架构



这里的数据结构需要在实际实现的时候与后端进行商量具体文档需要见：智胜的文档链接

交互难点及解决方案

拖拽逻辑

1. 需要保证 widget 不能重叠
2. 需要保证 在 column=16 的画布中 需要自动计算当前用量与剩余
3. 在向上插入 widget 时 保证 widget 会被自动向下排列，同时需要重新计算下面所有 widget 的排列方式

这里具体举几个例子需要考虑到

如 10*1 6*1 向上插入 11*1 则 11 需独占一行

如果插入的是 1*1 则 1*1 与 6*1 占领一行 同时递归计算 11*1 以及其下面的所有元素。

这里需要**碰撞检测和调整（Collision Detection and Adjustment）和 布局重新计算（Layout Recalculation）**

example:

```
  addItem(item) {
    this.items.push(item);
    this.autoPositionItem(item);
  }


  autoPositionItem(item) {
    for (let row = 0; row < this.rows; row++) {
      for (let col = 0; col < this.columns; col++) {
        if (this.canPlaceItem(item, row, col)) {
          this.placeItem(item, row, col);
          return;
        }
      }
    }
  }

  canPlaceItem(item, row, col) {
    for (let i = 0; i < item.rows; i++) {
      for (let j = 0; j < item.columns; j++) {
        if (this.grid[row + i]?.[col + j] !== null) {
          return false;
        }
      }
    }
    return true;
  }
```

无限扩展格子的实现与节流

我们不能一次在浏览器中画 很多格子 因为每个格子是一个元素浏览器当元素很多的时候会导致浏览器卡顿，所以必须做到动态加载格子

此处需要监听滚动条事件，如果滚动到底则加载 16*5 的格子量，同时不要加载可扩展格子这些格子只是虚拟格子它上面无法防止 widget

example:

```
  const {layoutRef} = props;
  const [items, setItems] = useState(Array.from({ length: 200 }));

  const handleScroll = () => {
    const { scrollTop, clientHeight, scrollHeight } = layoutRef.current || document.documentElement || document.
body;
    if (scrollTop + clientHeight >= scrollHeight - 20) {
      // Reached the bottom, load more items
      setItems(prevItems => [...prevItems, ...Array.from({ length: 80 })]);
    }
  };
```

动态扩展实际容积、左边栏卡片拖拽至右侧后的自动扩展

我们不允许 widget 在竖直方向有空行，如果有空行会导致页面极长无法提供好的用户体验。同时为了节流 我们需要动态计算当前 widget 的总高度 然后预制当前高度 row+1 个可放置 widget 的格子

那么由于上面的架构中讲到每个 widget 都有起相应的事件。所以我们需要设计一段程序，当 widget 被 added 的时候对外 trigger ，同时画布需要 listen 该事件将自己增加到总 row+1 的高度。

这种发布订阅者模式在 js 中很好实现

example:

```
  const addedRowTrigger = useCallback((grid: GridStack, listeners: Array<Function>) => {
    grid.on('added', function (event, items) {
      listeners.map((eventFunc) => {
        eventFunc(grid, items)
      });
    });
  }, []);
```

模板与实例的转换

要让拖进 dashboard 的 template 携带 template id
根据 templateid 自动找到相应的图

记录当前 id 点击确定的时候将相应的数据更新到该 widget 中
templateid 表明他是哪个组件
具体是哪个 widget 需要使用api

```
const clickWidgetTrigger = useCallback((grid: GridStack, listeners: Array<Function>) => {
    grid.on('click', function (event, items) {
      listeners.map((eventFunc) => {
        eventFunc(grid, items)
      });
    })
  }, []);
return {
    grid: grid.current,
    clickWidgetTrigger
}
```

这里要如法炮制一个 active widget 的事件监听机制，同时暴露出去，以便其他调用者监听

响应式解决方案

dashboard 的布局需要做到响应式所以我们使用了 flex 布局动态计算在一定的宽度下 分成 16 份且要求每份间隔为 10px

这里需要针对格子做自适应，因为是正方形的格子不可能采用断点的方式来实现，需要做到动态计算，为每个不同的像素做适配。

这里总体的计算方式是：

```
calc((100% - 150px) / 16);
```

```
calc((100% - 214px) / 16);
```

同时需要将格子的border 预留出来

可以使用 css 的属性 box-border规避border/padding 等属性

技术选型

库名	git hub	优点	周下载次数	更新频次	星	场景

react-grid-layout	https://www.npmjs.com/package/react-grid-layout	• **React集成：** 完全集成到React生态系统中，利用React的组件模型和状态管理。 • **简单易用：** 简单的API和直观的布局方式，易于入门和使用。 • **组件化：** 使用React组件构建布局，易于维护和扩展。 • **社区支持：** 作为一个受欢迎的React库，拥有强大的社区支持和文档。	432,535	4 days ago	18.7k	• BitMEX • AWS CloudFront Dashboards • Grafana • Metabase • HubSpot • ComNetViz • Stoplight • Reflect • ez-Dashing • Kibana • Graphext • Monday • Quadency • Hakkiri • Ubidots • Statsout • Datto RMM • SquaredUp
`gridstack.js`	https://github.com/gridstack/gridstack.js	• **高级布局：** 提供了更多高级布局和定制选项，适合构建复杂的网格布局。 • **交互性强：** 处理拖放、调整大小等交互行为，支持丰富的用户交互。 • **功能丰富：** 提供了丰富的功能和选项，包括嵌套网格、自定义小部件等。	123,072	7 days ago	5.5k	
`react-grid-item`						