✍️ 基于 draftjs 实现的计算表达式

一、背景

- 背景说明：当前数仓表尚未完善，仍需要用到 ods 的表同步实际数据
- 目标用户：目标管理员、分析师
- 业务场景：作为一个目标管理员，希望可以对源表提供的字段增加或剔除另一个字段的数据，获得一个复合使用需求的新字段
- 需求目标：支持用户自行构建计算指标，帮助用户更灵活的构建目标模板

二、需求说明

涉及公司内部文档，屏蔽

三、开发准备

3.1 开发前的准备工作

调研

立项：https://draftjs.org/

功能点：

1. 有效指标显示绿色，无效指标显示灰色，聚合函数显示蓝色
2. 选择聚合函数之后，光标自动定位到括号里面

3. 快捷删除，共计四种情况，backspace+],delete + [, [**backspace**],[**delete**]
4. 高度拖拽

3.2 进入开发阶段

根据 api 文档描述，可以根据 decorators 来设置不同的规则来实现功能点一

> # Decorators
>
> Inline and block styles aren't the only kind of rich styling that we might want to add to our editor. The Facebook comment input, for example, provides blue background highlights for mentions and hashtags.
>
> To support flexibility for custom rich text, Draft provides a "decorator" system. The tweet example offers a live example of decorators in action.

由于选择的指标根据数据的不同需要动态生成动态正则表达式

根据对应的正则表达式生成对应的装饰器，

```JavaScript
[指标] // 有效指标
[[abc]]// [abc]为有效指标
```

仅捕获最小匹配范围的中括号内部的指标，正则表达式采用**(?!pattern) 零宽负向先行断言 来捕获对应数据，捕获到的数据会应用到对应的 className**

```JavaScript
  const regExpList: Parameters<typeof
DraftEditor>[0]['regExpList'] = React.useMemo(
    () => [
      {
        // 匹配有效名字
```

```
        regExp: new
RegExp(`\\[((?!(\\[|\\]))(${validNames?.join('|')}))+\\]`, 'gi'),
        className: styles.indicator,
      },
      {
        // 匹配无效名字
        regExp: new
RegExp(`\\[((?!(\\[|\\]))(${unValidNames?.join('|')}))+\\]`,
'gi'),
        className: styles.disabled,
      },
      {
        // eslint-disable-next-line no-lookahead-lookbehind-
regexp/no-lookahead-lookbehind-regexp
        regExp: /\[((?!(\[|\])).)+\]/gi,
        className: styles.default,
      },
      {
        // 括号的数字
        regExp: new RegExp(`\\[((?!(\\[|\\]))\\d)+\\]`, 'gi'),
        className: styles.disabled,
      },
      {
        // 全局的数字
        regExp: new RegExp(`\\d+`, 'gi'),
        className: styles.indicator,
      },
    ],
    [validNames, unValidNames]
  );
```

```
JavaScript
 const plusDecorator = React.useMemo(() => {
    if (!regExpList) return [];
    return regExpList.map(item => ({
      strategy: (contentBlock, callback) =>
getStrategy(item.regExp, contentBlock, callback),
      component: ({ offsetKey, ...rest }: any) => (
        <MarkColor {...rest} className={item.className}
offsetKey={offsetKey} {...(item.extraProps ?? {})} />
      ),
    })) as CompositeDecoratorType;
  }, [regExpList]);
```

```JavaScript
// 原装装饰器
  const compositeDecorator = new
CompositeDecorator([...originDecorator, ...plusDecorator]);

  const [editorState, setEditorState] = React.useState(() =>
EditorState.createEmpty(compositeDecorator));
  useDraftPatch({
    editorRef,
    dispatchEditorState: setEditorState,
  });
  /*= ================================更新
==================================== */
  React.useEffect(() => {
    if (!value) {
      setEditorState(() =>
EditorState.createEmpty(compositeDecorator));
    } else {
      const createEditorWithText = () =>
EditorState.createWithContent(ContentState.createFromText(value),
compositeDecorator);
      updateEditor([createEditorWithText, moveFocusToEnd],
editorState);
    }
  }, [value]);
```

因此最后展示的功能点一即为

功能点二：

实现思路：由于需要定位光标到括号中，也就是在函数定位的时候，让光标放置在倒数第二位即可，因此聚合函数都是以 xxx()的形式存在

```JavaScript
// 更新选中坐标到括号
  const focusSelectionToBracket = (currentEditorState: EditorState) => {
    if (!currentEditorState) return currentEditorState;
    const selection = currentEditorState.getSelection();

    const focusOffset = selection.getEndOffset() - 1;
    // selection.merge()
    // console.info(focusOffset, data);
    return EditorState.forceSelection(
      currentEditorState,
      selection.merge({
        focusOffset,
        anchorOffset: focusOffset,
      })
    );
  };
```

功能点三：

[屏幕录制 2022-09-26 13.53.10.mov]

实现思路：由于 draftjs 并不提供现成的方式，但是提供了针对自定义键盘事件的处理方式，因此只要监听 backspace 和 delete 的事件，然后分别遍历前后寻找到的中括号来判断寻找到的字符串是否为已知指标（不管是有效指标还是无效指标，均为已知指标）

```
JavaScript
 const keyBindingFn: EditorProps['keyBindingFn'] = e =>
Draft.getDefaultKeyBinding(e);
  const handleKeyCommand: EditorProps['handleKeyCommand'] =
(command, currentEditorState: EditorState) => {
    const currentSelection = currentEditorState.getSelection();
    const currentContentState =
currentEditorState.getCurrentContent();
    // 判断当前编辑器是否被框选
    const isCollapsed = currentSelection.isCollapsed();
    const focusOffset = currentSelection.getFocusOffset();
    const text = currentContentState.getPlainText();
    // 如果被框选，直接按照框选删除的逻辑就行，不直接掌控
    if (!isCollapsed) return 'not-handled';

    const notHandled = (content: string) => {

      const matchValidResult = matchResultByExec(content,
validRegExp);
      const matchInvalidResult = matchResultByExec(content,
unValidRegExp);
      // 每次只删一个指标，长度大于1 说明匹配了多个指标
      if (matchValidResult.length + matchInvalidResult.length > 1)
return true;
      // 因此两个结果的和要么等于0 要么等于1
      if (matchInvalidResult.length) {
        const current = matchInvalidResult.pop();
        // 如果匹配的value 和当前捕获的content 长度相等，则需要被掌控
        if (current?.itemValue.length === content.length) return
false;
        return true;
      }
      if (matchValidResult.length) {
        const current = matchValidResult.pop();
        if (current?.itemValue.length === content.length) return
false;
        return true;
```

```
      }
      return true;
    };
    if (command === 'delete' || command === 'backspace') {
      if (text?.[focusOffset - 1] === ']') {
        if (command !== 'backspace') return 'not-handled';
        const leftBracketIndex = findLeft(focusOffset, text);
        if (leftBracketIndex === -1) return 'not-handled';

        const content = text?.slice(leftBracketIndex,
focusOffset);

        // 如果找到的指标不是在列表中的指标，不掌控
        if (notHandled(content)) return 'not-handled';
        // 如果找到了指标，将指标从编辑器实例中删除
        setEditorState(removeText(currentEditorState, focusOffset,
leftBracketIndex, 'backward'));
        return 'handled';
      }
      if (text?.[focusOffset] === '[') {
        if (command !== 'delete') return 'not-handled';
        const rightBracketIndex = findRight(focusOffset, text);
        if (rightBracketIndex === -1) return 'not-handled';
        const content = text?.slice(focusOffset,
rightBracketIndex);

        // 如果找到的指标不是在列表中的指标，不掌控
        if (notHandled(content)) return 'not-handled';
        setEditorState(removeText(currentEditorState,
rightBracketIndex, focusOffset, 'forward'));
        return 'handled';
      }

      const leftBracketIndex = findLeft(focusOffset, text);
      const rightBracketIndex = findRight(focusOffset, text);
      if (focusOffset <= leftBracketIndex || focusOffset >=
rightBracketIndex) return 'not-handled';
      if (leftBracketIndex === -1 || rightBracketIndex === -1)
return 'not-handled';
      const content = text?.slice(leftBracketIndex,
rightBracketIndex);

      // 如果找到的指标不是在列表中的指标，不掌控
      // 每次只删除一个指标
```

```
      // console.info(content);

      if (notHandled(content)) return 'not-handled';
      setEditorState(removeText(currentEditorState,
rightBracketIndex, leftBracketIndex, 'backward'));
      return 'handled';
    }
    return 'not-handled';
  };
```

功能点四：

```
JavaScript
 // 高度自由拉伸
 const [infoTabsDragInfo, infoTabsDragOnMouseDown, isMoving] =
useDraggable({ range: [0, 0, 0, Infinity] });
 <div
            style={{
              height: 80 + infoTabsDragInfo.deltaY,
            }}
          >
            //
          </div>
```

三、总结

总结本次技术应用中可积累的代码和经验

技术经验	有待提升

- 写下第一条经验
- ……

- 记录遇到的阻塞点
- ……